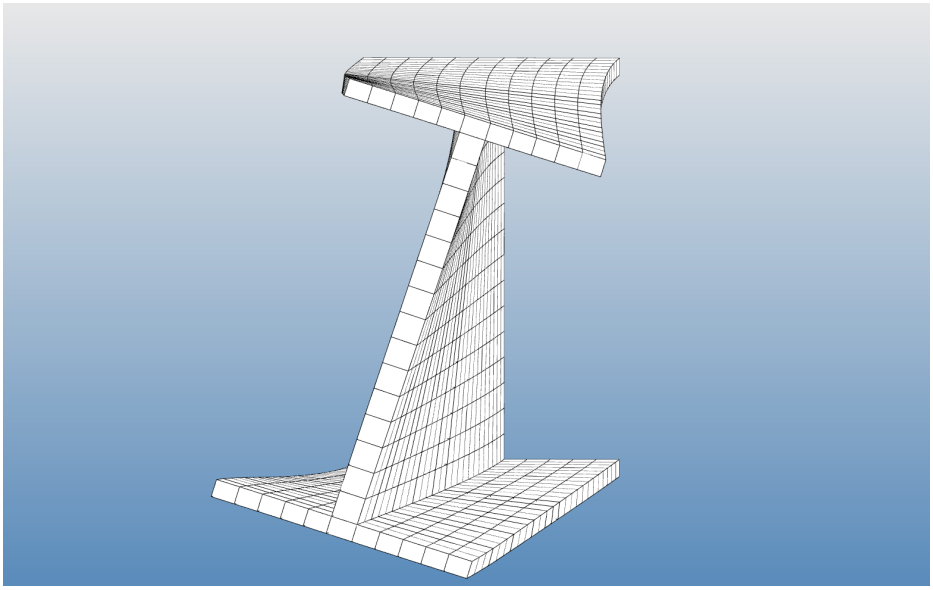


DYNAMISK PROGRAMVARA FÖR VISUALISERING AV VRIDNA PRISMATISKA BALKAR MED ÖPPNA, TUNNVÄGGIGA TVÄRSNITT

DAVID KINSELLA

Bachelor's Dissertation
at Structural Mechanics

DEPARTMENT OF CONSTRUCTION SCIENCES
DIVISION OF STRUCTURAL MECHANICS

ISRN LUTVDG/TVSM--14/6001--SE (1-88) | ISSN 0281-6679
BACHELOR'S DISSERTATION

DYNAMISK PROGRAMVARA FÖR VISUALISERING AV VRIDNA PRISMATISKA BALKAR MED ÖPPNA, TUNNVÄGGIGA TVÄRSNITT

DAVID KINSELLA

Supervisor: **JONAS LINDEMANN**, PhD, Div. of Structural Mechanics, LTH | LUNARC, Lund.
Examiner: Professor **PER JOHAN GUSTAFSSON**, Div. of Structural Mechanics, LTH, Lund.

For information, address:
Div. of Structural Mechanics, LTH, Lund University, Box 118, SE-221 00 Lund, Sweden.
Homepage: http://www.byggmek.lth.se

Förord

Denna rapport skrevs som kandidatarbete vid Lunds Tekniska Högskola, Avdelningen för Byggnadsmekanik. Uppkomsten av problemformuleringen till arbetet härstammar från diskussioner med prof. Per Johan Gustafsson. Jag skulle vilja tacka tekn. dr Jonas Lindemann på LUNARC i Lund för handledningen. Tack också till arkitektstudenten Svetlana Tutanova för kreativa synpunkter.

Lund, Allhelgonaafton 2014

David Kinsella

Abstract

Reportedly in Swedish schools, there is no long tradition of investigative and laboratory work in the mathematics subject. Dynamic software is a term which encapsulates the purpose of software as a pedagogical instrument in teaching. There is great potential in this but also a need for teachers to engage and apply themselves to the new possibilities. Computer software in class does not automatically generate good results. As an example of how software can be developed in Python to aid in the learning of torsional beam theory, a user-friendly application is developed for 3D visualisation of a twisted beam element and for the determination of the maximal cross-sectional tensions and rotations. The software can e.g. be used by a student of mechanics as a pedagogical instrument, seeing as the theory itself is rather complex, in particular when it comes to warping constraints. Forming a base for the software are a number of open source Python bindings and modules: Visvis, NumPy, PyCALFEM and PyQt. The visualized beam element can be rotated, moved around and panned in the GUI with simple mouse clicks and keyboard actions. The 3D shape of the beam element is calculated with the FEM. Three types of open, thin-walled cross sections are available to the user: simply symmetrical I-profiles, symmetrical U-profiles with partially constant wall thickness and polar symmetrical Z-profiles with partially constant wall thickness. The material is modelled as linear elastic and the beam is either 1) freely supported with forks in each end or 2) fully fixed in each end or 3) a console. The maximal cross sectional tensions and rotations are further calculated according to St Venant's and Vlasov's beam theory and solutions to these are approximated with the FEM. The application is free to use for educational purposes.

Sammanfattning

Skolan har ingen lång tradition av att arbeta laborativt i matematik och dynamisk programvara är ett sätt att fylla denna brist. Dynamisk programvara låter eleven gå i dialog, utmanar, skapar nyfikenhet, åskådliggör begrepp och underlättar inlärningen. Det krävs emellertid att lärare är villiga att sätta sig in i de relativt nya möjligheterna eftersom datoranvändning i matematik- och fysikundervisningen inte per automatik genererar goda resultat. Ett användarvänligt program för visualisering i 3D av en vriden balk och för beräkning av maximala spänningar och rotationer i tvärsnittet utvecklas i programmeringsspråket Python. Användaren kan så att säga ställa frågor till programmet: "Vad händer om jag gör så här?" och spörsmål utredas. Programmet bygger på en rad paket och moduler baserade på öppen källkod: Visvis, NumPy, PyCALFEM och PyQt. Den visualiserade balkkroppen kan vridas och vändas, flyttas och panoreras med enkla musklick och tangenttryck. 3D-figurens deformationer beräknas med FEM. Tre sorters öppna, tunnväggiga tvärsnittsprofiler är tillgängliga: enkelsymmetriska I-tvärsnitt, symmetriska U-tvärsnitt med styckevis konstant väggtjocklek och polärsymmetriska Z-tvärsnitt med styckevis konstant väggtjocklek. Materialet är linjärt elastiskt och balken antingen 1) gaffellagrad i båda ändar eller 2) fast inspänd i båda ändar eller 3) en konsol. Maximala spänningar och rotationer i tvärsnittet beräknas enligt S:t Venants och Vlasovs teori och lösningar approximeras med FEM. Programmet är fritt tillgängligt i pedagogiskt syfte.

Innehåll

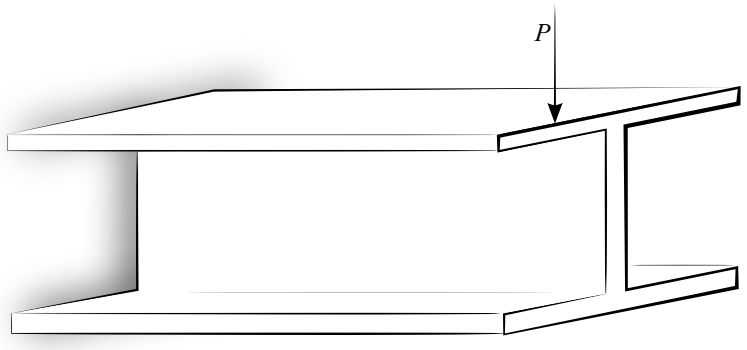
1 Inledning

Vridning i statiska konstruktionsdelar orsakas oftast av lastexcentricitet eller geometriska imperfektioner och den vanligaste förekomsten är i raka eller krökta fackverk [5]. Vridning kan också förekomma under tiden som en konstruktion uppförs. För att förhindra detta brukar avstyvningar, stag och stöd användas.

Betrakta konsolbalken i Figur 1.1. Antag att lasten angriper excentriskt med hänsyn till skjuvcentrum så att vridning inträffar. Hur kan en elev som studerar fysiken bakom balkvridning visualisera den rotation som sker i tvärsnittet på grund av den excentriska lasten? Hur påverkas rotationen om ett annat tvärsnitt väljs? Hur ser välvningen ut för olika sorters geometrier på tvärsnittet? Hur stora blir spänningarna orsakade av vridningen?

Å ena sidan kan svaren sökas i de analytiska uttrycken. S:t Venantsk vridning ger rotationen som en funktion av vridstyvhetens tvärsnittsfaktor under förutsättning om fri välvning. Vlasovsk vridning ger rotationen som en funktion av välvstyvhetens tvärsnittsfaktor, den normerade sektoriella koordinaten och det statiskt sektoriella momentet.

Å andra sidan kunde ett balkelement som utsätts för vridning visualiseras i 3D med hjälp av FEM och vridning och välvning studeras därigenom. Eleven skulle kunna vrida och vända på balken i 3D, ändra på geometriparametrar, ändra lasten, ändra upplagsförhållandena och så vidare. En programvara kunde tas fram för visualiseringen och användaren kunde så att säga ställa frågor till programmet: “Vad händer om man gör så här. . . ” Det här undersökan-



Figur 1.1: Excentrisk belastning på konsolbalk ger upphov åt vridning.

de och visuella arbetssättet skulle kunna öka förståelsen hos en nybörjare för vridnings- och välvningsfenomen på ett helt annat sätt än studiet av de analytiska uttrycken allena skulle göra.

Den här rapporten skall illustrera hur en datoranvändare (till exempel en lärare) med relativt enkla medel kan skapa ett fullständigt användargränssnitt baserat på öppen källkod för visualisering i 3D av vridna balkar och beräkning av förekommande spänningar och rotationer. Programmet kan hjälpa den som studerar balkvridning att varsebli fenomen såsom välvning och tjäna som stöd i studiet av den bakomliggande teorin. Programmet kan också fungera som inspiration för den som önskar jobba vidare med dynamiska datorprogram och är beredd att lägga ner tid på utvecklingen av ett sådant för skräddarsydda behov. De metoder och principer som presenteras i rapporten kan anpassas för att visualisera och begreppsliggöra även andra fysikaliska fenomen beroende på vilka differentialekvationer som ligger till grund, exempelvis elektriska fält, vätskeflöden, värmeledning, akustik med mera.

1.1 Syfte

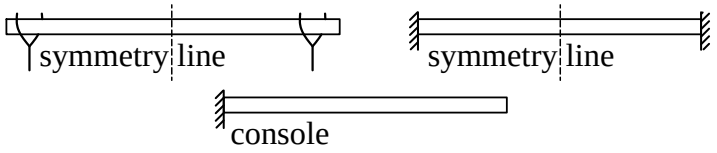
Syftet med rapporten är att undersöka hur prismatiska balkar med tunnväggigt tvärsnitt kan visualiseras med öppen källkod och hur programvara kan tjäna som ett didaktiskt redskap och stöd i matematik- och fysikundervisningen.

1.2 Frågeställningar

- Hur kan datorprogram användas i matematik- och fysikundervisningen för att stödja inlärningen och främja ett undersökande arbetssätt och vilken vetenskaplig grund finns för detta?
- Hur görs ett gränssnitt med öppen källkod för visualisering och beräkning i 3D av vridna balkar?
- Vilken teori ligger till grund för visualiseringen och beräkningen?

1.3 Avgränsningar

- Endast prismatiska balkar studeras
- Balken är antingen 1) gaffellagrad i båda ändar eller 2) fast inspänd i båda ändar eller 3) en konsol. De tre randvillkoren illustreras i Figur 1.2.
- Vridmomentet är antingen jämnt fördelat utmed balkens längd eller annars punktformigt i balkmitt eller i balkänden för fallet med konsolen.
- Tre slags öppna, tunnväggiga tvärsnitt undersöks: enkelsymmetriska I-tvärsnitt, symmetriska U-tvärsnitt med styckevis konstant väggtjocklek



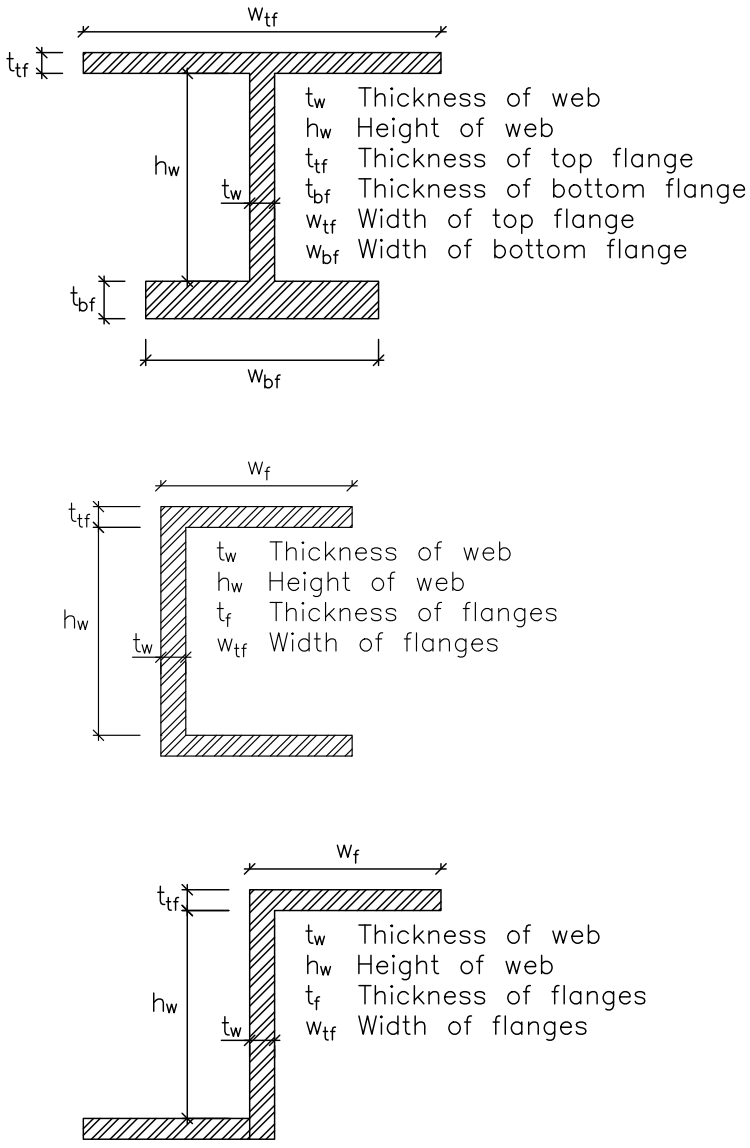
Figur 1.2: Balken är antingen 1) gaffellagrad i båda ändar eller 2) fast inspänd i båda ändar eller 3) en konsol.

samt polärsymmetriska Z-tvärsnitt med styckevis konstant väggtjocklek. Tvärsnitten illustreras i Figur 1.3 tillsammans med tillhörande mått.

- Materialet är isotropt och linjärelastiskt.

1.4 Tidigare forskning

Lil Engströms avhandling om möjligheter till lärande i matematik genom dynamisk programvara samlar en hel del av den tidigare forskningen om datorstödd undervisning utifrån ett svenskt perspektiv och redovisar också den historiska datorsatsningen i den svenska skolan [7]. Engströms avhandling ger en god grund att stå på för den som önskar sätta sig in i den svenska forskningen om dynamisk programvara. Högskoleverket har gett ut en skrift som samlar intryck och erfarenheter från datoranvändningen vid en rad svenska universitet och tekniska högskolor, bland annat Lunds Tekniska Högskola, Chalmers Tekniska Högskola, Luleå Tekniska Universitet och Kungliga Tekniska Högskolan [13]. Wolfgang Christian och Mario Belloni vid Davidson College i USA har varit poinjärer när det gäller utvecklingen av gratis programvara i Java som stöd i fysikundervisningen [3]. Deras attityd och inställning till datoranvändningen har varit inspirerande. Steve VanWyk har ett mycket detaljerat och intressant bibliotek av Mathematica-kod för visualisering av en rad vanligt förekommande fysikaliska fenomen [41]. Minna Salminen-Karlssons avhandling om genuskontrakt vid datorteknikutbildningar i Sverige problematiserar bilden av den stereotype datorprogrammeraren såsom manlig och förmågan att datorprogrammera såsom en exlusivt manlig kompetens [31]. Från siffror till surfning är en antologi som anlägger ett könsperspektiv på bland annat numerisk analys [15]. Den grundläggande balkteorin återfinns i [11]. En masteruppsats om numeriska metoder för balkvridning skriven av en student på Massachusetts Institute of Technology år 1997 har varit inspirerande [5]. Hans arbete saknar emellertid visualisering i 3D och är inte förankrat i det moderna och användarvänliga Python-språket. Teorin för finita elementmetoden (FEM) återfinns i Ottosens referensverk [25]. Python, PyQt, NumPy och Visvis har paket på nätet med vars hjälp det utmärkt går att lära sig hur dessa fungerar [29, 27, 23, 42]. Användningen av PyCALFEM framgår av bland annat av Andreas Edholms masteruppsats från 2013 skriven vid Avdelningen för Byggnadsmekanik på Lunds Tekniska Högskola [6] och Andreas Ottossons uppsats från samma institution [26] år 2010. Därutöver finns ett flertal exempelskript i PyCALFEM-biblioteket att tillgå. Den som önskar sätta sig in i CALFEMs funktionalitet kan vända sig till CALFEM-manualen [1].



Figur 1.3: Tre slags öppna, tunnväggiga tvärsnitt undersöks: enkelsymmetriska I-tvärsnitt, symmetriska U-tvärsnitt med styckevis konstant väggtjocklek samt polärsymmetriska Z-tvärsnitt med styckevis konstant väggtjocklek.

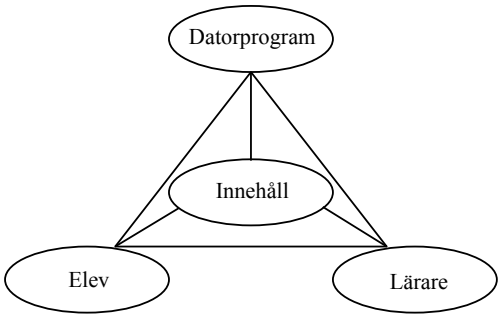
2 Datorstödd undervisning

2.1 Inledning

Skolan har ingen lång tradition av att arbeta undersökande och laborativt i matematik, hävdar Lil Engström i sin avhandling om möjligheter till lärande i matematik [7]. Vad som saknas är den sortens undersökande arbetssätt där matematiska begrepp och samband kan upptäckas och hypoteser formuleras. Samma slutsats dras i en av statens offentliga utredningar som hävdar att det råder brist på möjligheter för eleven att spekulera, skapa, argumentera och verifiera [13]. Samtidigt finner Engström att lärare ofta träder tillbaka inför den nya och potentiellt laborativa miljö som datorprogramvara utgör med resultatet att eleverna lämnas åt sig själva. Ibland kan det såklart bero på bristande erfarenhet och kompetens hos den undervisande läraren. Forskning visar att ju mer välutbildad en matematiklärare är, desto mer flexibel är hen i sättet att använda datorprogram i undervisningen [7]. Lärarens kompetens har i det här sammanhanget en avgörande betydelse för elevernas inlärningsresultat.

2.2 Dynamisk programvara

Datorprogrammets position i undervisningen kan illustreras med hjälp av Klafkis didaktiska triangel, se Figur 2.1. Datorprogrammet är inte bara en del av undervisningens innehåll utan framförallt ett didaktiskt redskap som kan utmana eleverna och skapa nyfikenhet, underlätta inlärningen och öka förståelsen av



Figur 2.1: Modell av samspelet mellan lärare, elev och datorprogram i en viss undervisningsmiljö. Bygger på Klafkis didaktiska triangel [7].

olika samband inom matematiken. Datorprogrammet kan enligt Säljö fungera som en kommunikativ partner som stödjer elevens tänkande [34]. Eleven ställer frågor till programmet, påverkas av det som syns på skärmen och leds att ställa nya frågor och så vidare. Datorn blir en katalysator som genererar frågor av slaget: "Vad händer om. . . ". Frågorna verkställs genom kommandon till datorn (musklick, tangentbordstryck odylikt). Engström benämner den här sortens datorprogram för *dynamisk programvara* i motsats till benämningen *interaktiv programvara* som hon finner är alltför begränsad i betydelse [7]. Dynamisk programvara leder således bort från en styrd och traditionell läromiljö till en öppen och laborativ sådan. Det är skillnaden mellan "matematik *för* problemlösning" och "matematik *genom* problemlösning", för att tala med Kerstin Ekstig [13]. Innebörden av det förstnämnda är snarast det tradionsbundna inövandet av teori följt av tillämpningar och problemlösning medan det sistnämnda omfattar det utforskande arbetssättet med (till exempel) dynamisk programvara.

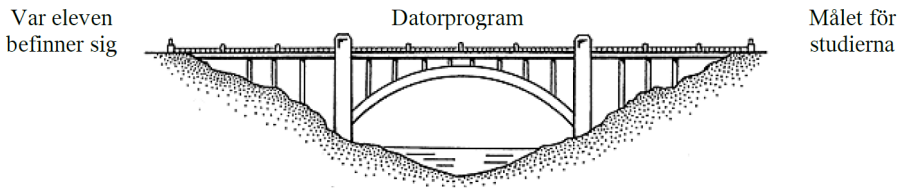
2.3 Kunskapssyn

Skillnaden mellan det eleven klarar av på egen och det den klarar av med stöd benämns av Lev Vygotskij[1] för den närmaste utvecklingszonen ("Zone of proximal development") [8]. Välanpassad programvara kan vara en stödstruktur, *scaffolding* för att använda Vygotskijs benämning, som hjälper eleven i den närmaste utvecklingszonen. Utgångspunkten i Vygotskijs syn på inlärning är människan som en sociokulturell varelse. Inlärning av komplexa färdigheter bygger enligt honom på förmågan att hantera olika slags medierande redskap. Penna, papper och dator är exempel på fysiska medierande redskap, så kallade *arketyper*, medan alfabetet, siffersystem och klassifikationer är exempel på psykologiska medierande redskap. Det förnämsta medierande redskapet enligt Vygotskij är det mänskliga språket som ligger till grund inte bara för den mellanmänskliga kommunikationen utan också för den interna kommunikation en människa ägnar sig åt genom sitt tänkande. Kunskap är något som approprieras i en process då en person använder (och lär sig använda) de medierande redskapen i en social kontext [8].

Vygotskijs kunskapssyn står i motsatsförhållande till synen på kunskap som diskreta "paket" som traderas från en lärare eller institution. Kunskap vinner en person snarare genom dialog, utforskande arbetssätt och formativ bedömning i vilken friheten att ha fel används konstruktivt för att ställa nya frågor i en ständigt fortskridande process (jämför även [18]). Här ingår att tolka det observerade och i bästa fall dra slutsatser och generalisera. Dynamisk programvara enligt Engström [7] rimmar väl med en sociokulturell kunskapssyn.

Flera undersökningar visar att datorn används mestadels till ordbehandling, faktasökning på Internet och som ett presentationsverktyg i skolan [13]. För att datorn skall fungera som ett dynamiskt, didaktiskt verktyg i matematik- och fysikundervisningen krävs mer än så. Enligt Anders Tengstrand [13] kan datorn användas i undervisningen på ett av tre sätt

[1] Lev Semënovic Vygotskij (1896-1934) var rysk psykolog och grundare av den sociokulturella teorin.



Figur 2.2: Datoranvändningen bör vara ett sätt att överbrygga gapet mellan den punkt där eleven befinner sig nu och den punkt som är målet för studierna.

- som räknehjälpmedel
- för att skapa förståelse
- som experimentverktyg för att upptäcka samband

Wolfgang Christian och Mario Belloni [3] anser att datorsimuleringar måste vara autentiska, genomförbara och anpassningsbara. För Christian & Belloni innebär detta först och främst att läraren har kompetensen att skräddarsy programvara åt sina elever och de visar också hur detta kan göras i programmeringsspråket Java. I föreliggande rapport visas hur ett gränssnitt kan utvecklas i Python. Jämfört med Java är Python ett ännu mer användarvänligt och läsbart språk, se vidare under avsnittet Python i Kapitel 4. Naturligtvis ger datoranvändning i undervisningen inte resultat automatiskt. Kerstin Ekstig frågade en gång om undervisningen vore dator*stödd* eller dator*störd* [13]. Datoranvändningen bör vara ett sätt att överbrygga avståndet mellan den punkt där eleven befinner sig för tillfället, och den punkt som motsvarar målet för studierna, se Figur 2.2. Detta kan jämföras med begreppet formativ bedömning som enligt Lundahl [18] kan sägas "peka ut mål och ange var i lärprocessen eleven befinner sig". När lärandemålen, undervisningen och bedömningen länkas samman och synliggörs för eleven fås en linje som i litteraturen benämns "alignment" [18]. Formativ bedömning rimmar väl med John Deweys[2] kunskapssyn. Kunskap är enligt honom inte bara ett mål eller ett resultat utan framför allt en process [12]. Dewey betonar också vikten av kommunikation, återkoppling och reflektion för all inlärning och den formativa bedömningen bygger i mångt och mycket på sådana medel [8]. Mer allmänt så kan man relatera formativ bedömning till den progressiva traditionen som Dewey var en del av. Se vidare Lundahl [18] för praktiska råd om hur formativ bedömning kan införlivas i undervisningen.

Det är uppenbart att det krävs god planering från lärarens sida för att framgångsrikt använda datorn i undervisningen. Datorprogrammen skall vara så enkla att använda att det inte tar någon tid från undervisningen. Eleven ska från början kunna koncentrera sig på den matematiska problemlösningen. Det finns också en risk att eleven inte skall klara enkla rutinuppgifter för hand om denne endast jobbar med datorn. Då blir förståelsen lidande. Man måste också kunna räkna för hand med papper och penna. Lärarens uppgift blir att hitta en lämplig avvägning. [13]

[2]John Dewey (1859-1952) var amerikansk filosof och psykolog.

2.4 Genusperspektiv

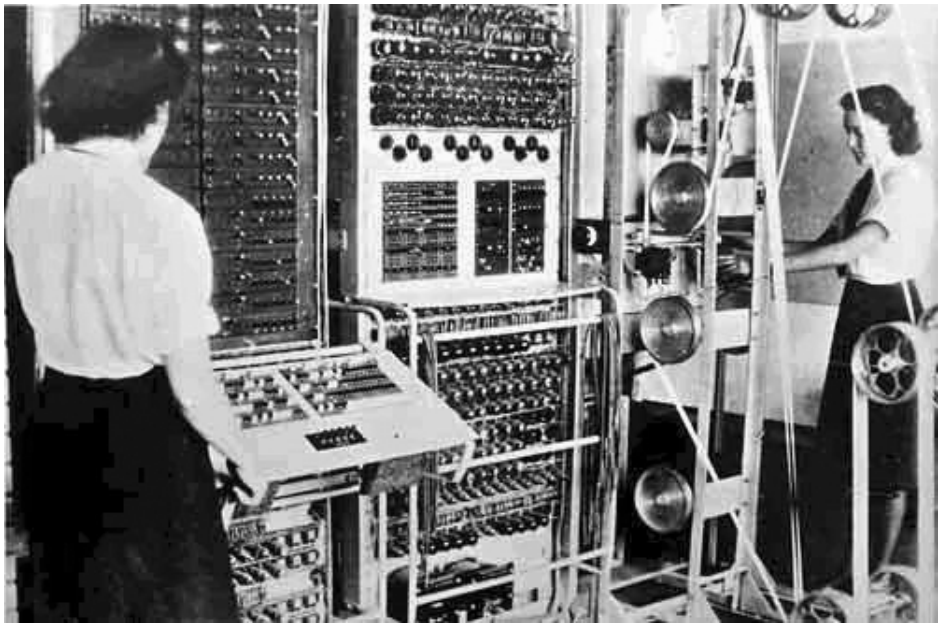
Enligt Minna Salminen-Karlsson [31] är datorn som artefakt inte direkt genuskodad. Vidare är datorprogrammering en tämligen abstrakt aktivitet och huruvida det skall uppfattas som manligt eller kvinnligt beror snarare på vem som använder datorn (eller programmerar) och på hur det görs. Maria Nordström och Gunilla Wikström [15] medger att "[d]et är mycket svårt att tänka sig den rena matematiken som speciellt maskulin" men försöker i en artikel med titeln "Kan man anlägga ett könsperspektiv på numerisk analys" göra gällande att beräkningsprogrammering vore tämligen maskulint genom ett *ad hominem*-argument: eftersom vetenskapen enligt feministisk analys är maskulint konstruerad blir också dess produkter såsom datorn och beräkningsprogrammeringen det. Mot detta synsätt kan som redan nämnts ställas Salminen-Karlssons ståndpunkt. Det finns hur som helst ett sammanhang kring datorn som trots allt leder till att kvinnor har utestängts och fortfarande, enligt Salminen, drar sig för att sysselsätta sig med datorteknik. Frågan väcks därför om genus i samband med dynamisk programvara i undervisningssyfte. Genus kan med Salminen-Karlsson [31] definieras som en rad kännetecken vilka konstrueras och tilldelas i en social praktik. Genus skapar skillnader mellan män och kvinnor som inte har med natur, essens eller biologi att göra.

Kvinnor har alltid varit bland dem som använt datorer. Det engelska ordet "computer" härstammar från den till övervägande del kvinnliga stab av (hand)räknare som betjänade de tidiga matematiska statistikerna (såsom Karl Pearson[3]) med siffror vid förra sekelskiftet [35]. Emellertid har kvinnornas uppgift historiskt sett handlat om att sköta om maskinerna, inte att programmera eller designa dem, även om det finns undantag. Jämför även Figur 2.3 som visar två kvinnor som sköter om *Kolossen*, den första digitala datorn från 1943 som användes av britterna för att knäcka tysk kryptering under andra världskriget [40]. Tänkbara anledningar till att kvinnor uteslutits från utvecklingen av datorteknik inbegriper enligt Salminen [31]

- datorer utvecklades för militära syften i försvarsorganisationer som många kvinnor inte hade tillgång till
- datorer utvecklades också på universitet av akademiker såsom matematiker och ingenjörer av vilka de flesta var män
- i 1950-talets USA då datorutvecklingen tog fart var den förhärskande ideologin i medelklassen att en familj skulle försörjas av en man medan en kvinna var hemmafru

Samtidigt finns det en viss grupp av män som ser datorn och programmering som en chans att klättra i den manliga hierarkin. Det rör sig om intellektuella män som saknat den fysiska styrka som vanligtvis förknippas med manlighet [31]. Datorn blir för dem ett synligt och teknologiskt uttryck för manlighet som det blir nödvändigt att försvara och bevaka. En strategi för att utestänga andra yrkesgrupper och kategorier från att lägga beslag på statusen inbegriper

[3]Karl Pearson (1857-1936) var engelsk matematiker, biometriker och grundare av avdelningen för matematisk statistik vid University College London.



Figur 2.3: Två kvinnor sköter om *Kolossen*, den första digitala datorn från 1943 som användes av britterna för att knäcka tysk kryptering under andra världskriget.

upprätthållandet av en esoterisk image kring datorteknik och mystifierandet av datorprogrammering genom exempelvis hacker-kulturer som typiskt sysselsätter sig med hjältetematik. Detta har såklart slagit mot kvinnor som grupp betraktat även om kvinnor inte varit ett exklusivt mål.

3 Balk- och beräkningsteori

3.1 S:t Venantsk vridning

Vridning enligt S:t Venant[1] bygger på antagande om elasticitetsteori, det vill säga små förskjutningar och töjningar och noll normalspänningar svarande mot fri välvningsdeformation av balktvärsnitt [11]. Av detta följer att spänningarna är proportionella mot töjningen. Rotationen av en materialpunkt i tvärsnittet är proportionell mot punktens avstånd till vridcentrum [5]. För en balk av isotropt[2] linjärt elastiskt material sammanfaller skjuvcentrum och vridcentrum [11]. Balkens vridningsaxel är en linje genom vridcentrum i tvärsnittet. Figur 3.1 illustrerar detta. S:t Venants teori behandlar skjuvtöjningar som leder till välvning. Sambandet mellan vridmomentet T och rotationen φ eftersöks. Lägg märke till att T kan uttryckas i skjuvspänningarna τ_{xz} och τ_{xy} genom statisk ekvivalens,

$$T = \int_A (\tau_{xz} y - \tau_{xy} z)\,\mathrm{d}A \tag{3.1}$$

Jämför Figur 3.2. Betrakta nu differentialelementet i Figur 3.3. Kraftjämvikt i x-led ger

$$\frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} = 0 \tag{3.2}$$

Med hjälp av momentjämvikt runt y- och z-axlarna inses att

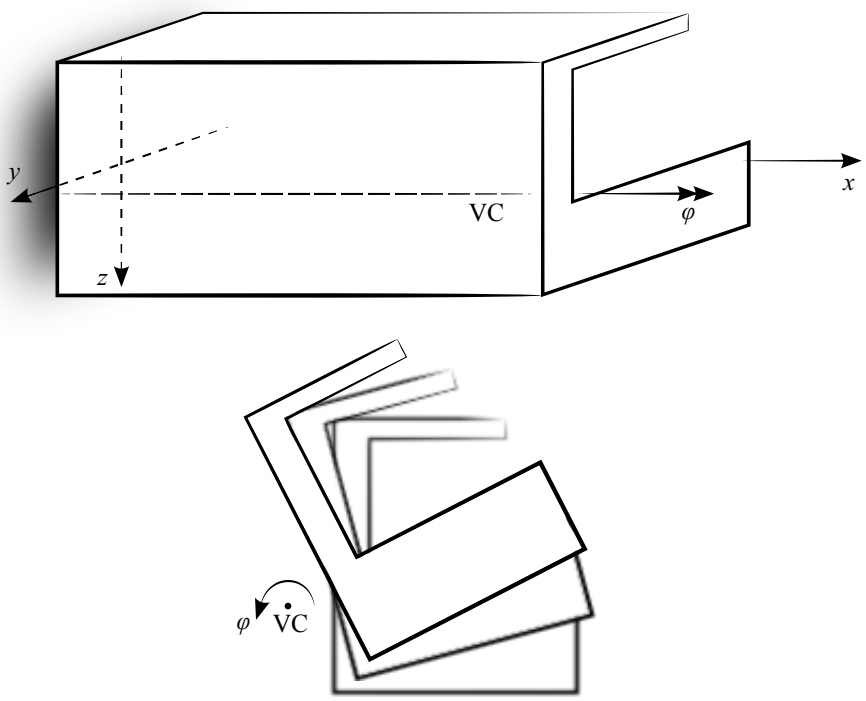
$$\begin{aligned} \tau_{xy} &= \tau_{yx} \\ \tau_{xz} &= \tau_{zx} \end{aligned} \tag{3.3}$$

Antag att origo är i vridcentrum. Inför ett samband mellan vridning och förskjutning (i tvärsnittet) enligt [5]

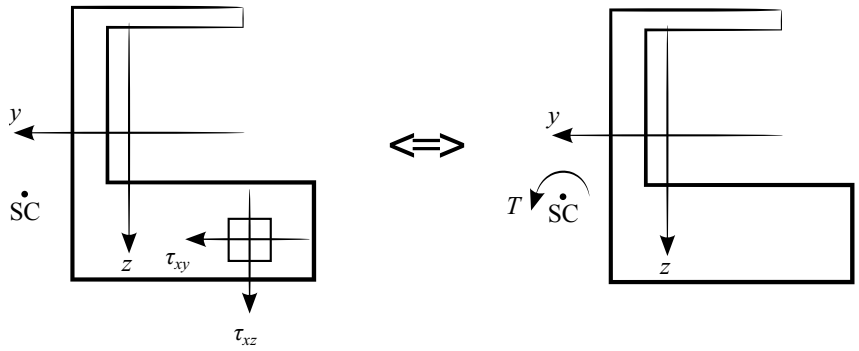
$$\begin{cases} v(x,y,z) = -z\varphi \\ w(x,y,z) = y\varphi \end{cases} \tag{3.4}$$

[1] Adhémar Jean Claude Barré de Saint-Venant (1797-1886) var fransk fysiker och matematiker.

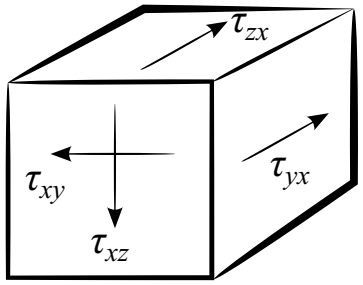
[2] Skjuvcentrum och vridcentrum överensstämmer också för linjärt elastiska ortotropa material.



Figur 3.1: Vridcentrum för en prismatisk balk av isotropt linjärt elastiskt material.



Figur 3.2: Statisk ekvivalens för en positiv snittyta



Figur 3.3: Differentialelement i balken

och utnyttja definitionen av skjuvtöjningar

$$\begin{cases} \gamma_{xz} = \dfrac{\partial u}{\partial z} + \dfrac{\partial w}{\partial x} \\ \gamma_{yz} = \dfrac{\partial v}{\partial z} + \dfrac{\partial w}{\partial y} \end{cases}$$

Det följer då av (3.4), (3.5) och linjärelasticiteten att (3.2) kan omskrivas

$$\frac{\partial \tau_{xy}}{\partial y} - \frac{\partial \tau_{xz}}{\partial z} = -2G\frac{\mathrm{d}\varphi}{\mathrm{d}x} \tag{3.5}$$

eftersom linjärelasticiteten innebär att $\gamma = \frac{\tau}{G}$ där G är materialets skjuvmodul [11]. Sambandet mellan rotationen φ och momentet T ges nu av ekvationssystemet bestående av (3.1) och (3.5). Systemet är emellertid underbestämt på grund av de tre obekanta τ_{xy}, τ_{xz} och φ. För att reducera antalet obekanta används Airys spänningsfunktion. Då fås ett samband mellan skjuvspänningarna. Antag därför att det finns en funktion Ψ (Airys spänningsfunktion) sådan att

$$\tau_{xy} = \frac{\partial \Psi}{\partial z} \qquad \tau_{xz} = -\frac{\partial \Psi}{\partial y} \tag{3.6}$$

Då uppfyller Ψ kraftjämvikten (3.2) och är en potentialfunktion över potentialfältet $(\tau_{xy}, -\tau_{xz})$ [21]. Insättning i (3.5) ger Poissons ekvation

$$\frac{\partial^2 \Psi}{\partial y^2} - \frac{\partial^2 \Psi}{\partial z^2} = -2G\frac{\mathrm{d}\varphi}{\mathrm{d}x} \tag{3.7}$$

Å andra sidan ger insättning av (3.6) i (3.1) att

$$T = -\int_A \left(\frac{\partial \Psi}{\partial y} y + \frac{\partial \Psi}{\partial z} z \right) \mathrm{d}A$$

som med Greens formel blir

$$T = 2\int_A \Psi \, \mathrm{d}A \tag{3.8}$$

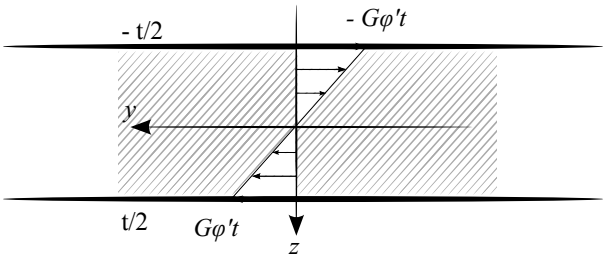
förutsatt att Ψ är harmonisk [5]. Sambandet mellan vridmomentet T och rotationen φ ges nu av (3.7) och (3.8).

Prandtl var först med att inse likheten mellan Airys spänningsfunktion och utböjningen av en tunn hinna. Analogin benämns även såphinneanalogin därför att spänningsfunktionen över tvärsnittsytan antar samma form som en ideal såpbubbla skulle göra, förutsatt att utbuktningen av såbubblan är liten. En hinna kännetecknas av att den är tunn i förhållande till sin utsträckning så att det förenklat kan antas att endast axiella krafter råder. Analogin medför att hinnans lutning är proportionell mot skjuvspänningen som uppstår på grund av vridningen av balken. Momentet är dubbelt så stort som volymen under spänningsfunktionen [11].

Ett slutet uttryck för sambandet mellan vridmomentet T och rotation φ kan emellertid inte tas fram med mindre än att spänningsfunktionen specificeras.

Tabell 3.1: Analogitabell S:t Venantsk vridning och stångverkan

Stångverkan	u	N	q_x	EA	L
Vridning	φ	T	q_v	GK_v	L



Figur 3.4: Skjuvspänningsfördelning för ett tunnväggigt rektangulärt tvärsnitt.

Eftersom materialet är linjärt elastiskt är vridmomentet T proportionellt med skjuvmodulen G och förvridningen $\frac{\mathrm{d}\varphi}{\mathrm{d}x}$. Proportionalitetsfaktorn K_v beror av tvärsnittets geometri och definieras av sambandet

$$T = G\,K_v\,\frac{\mathrm{d}\varphi}{\mathrm{d}x} \tag{3.9}$$

Vanligen benämns K_v *vridstyvhetens tvärsnittsfaktor* [11]. Om balken är belastad med ett fördelat vridmoment q_v ger vridjämvikt runt x-axeln

$$\frac{\mathrm{d}T}{\mathrm{d}x} + q_v(x) = 0 \tag{3.10}$$

Ekvationerna (3.9) och (3.10) ger tillsammans grundekvationen för vridning

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(GK_v\frac{\mathrm{d}\varphi}{\mathrm{d}x}\right) + q_v = 0 \tag{3.11}$$

Lösningar söks med angivande av de statiska och kinematiska randvillkoren. Eftersom grundekvationen (3.11) är helt analog med grundekvationen för en axialbelastad stång, kan ett S:t Venantskt torsionselement i FEM fås genom direkt användning av styvhetsmatrisen för stångverkan. De analoga storheterna anges i Tabell 3.1.

Vridstyvhetens tvärsnittfaktor

För cirkulära tvärsnitt kan K_v beräknas exakt genom en väl vald ansättning. För en tunnväggig balk med rektangulärt tvärsnitt (bredd b, tjocklek t och $t << b$) går det att visa att [11]

$$\Psi = G\frac{\mathrm{d}\varphi}{\mathrm{d}x}\left(\left(\frac{t}{2}\right)^2 - z^2\right)$$

så att

$$K_v = \frac{bt^3}{3}$$

För andra tvärsnittsformer föreslår Kollbrunner & Basler ([16] genom [5])

$$\beta I_r K_v = A^4$$

där β är en konstant som avspeglar tvärsnittsytans särart, I_r är det polära yttröghetsmomentet och A tvärsnittsytan. För ett rektangulärt tvärsnitt med $t << b$ fås exempelvis

$$\beta = \frac{A^4}{I_r K_v} = \frac{tb^4}{\frac{tb^3}{12} \cdot \frac{bt^3}{3}} = 36$$

Ett cirkulärt tvärsnitt ger å andra sidan $\beta \approx 40$. För ett tunnväggigt rektangulärt tvärsnitt är spänningarna riktade i x-led enligt Figur 3.4 och den maximala skjuvspänningen

$$\tau_{max} = G\frac{\mathrm{d}\varphi}{\mathrm{d}x}t$$

Det så kallade *vridmotståndet* definieras av [11]

$$\tau_{max} = \frac{T}{W}$$

För tvärsnitt sammansatta av flera tunna rektanglar går det att visa att

$$\begin{aligned} K_v &= \sum_{i=1}^{n} \frac{b_i {t_i}^3}{3} \\ W &= \frac{K_v}{t_{max}} \end{aligned}$$

3.2 Vlasovsk vridning

Vridning enligt Vlasov[3] bygger på antagandena

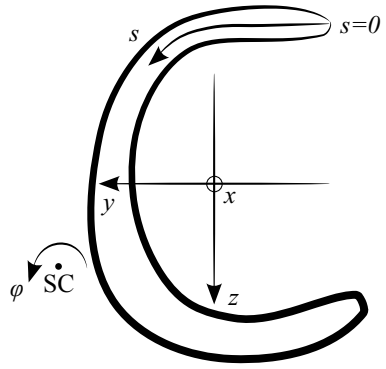
- tvärsnitten deformeras ej i yz-planet
- tvärsnitten roterar kring skjuvcentrum (SC)
- skjuvtöjningarna försummas

Punkterna i tvärsnittet beskrivs av en koordinat s i ett kroklinjigt koordinatsystem, se Figur 3.5. Väggtjockleken $t(s)$ kan bero på s. Skjuvtöjningen är

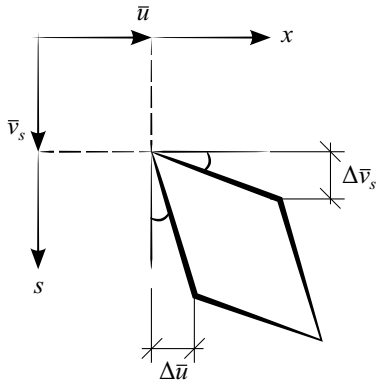
$$\frac{\partial u}{\partial s} + \frac{\partial v_s}{\partial x} = \gamma_{xs} \tag{3.12}$$

vilket illustreras i Figur 3.6 under förutsättning att vinkeländringarna är små så att $\sin x \approx x$ [11].

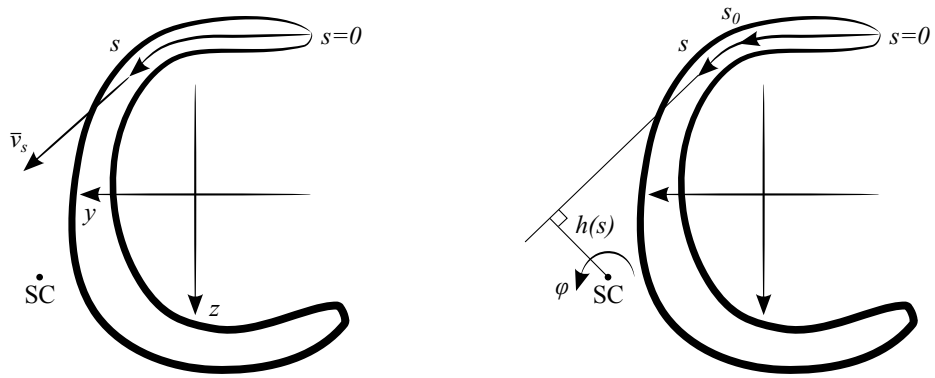
[3]Anatoly Alexandrovich Vlasov (1908-1975) var rysk teoretisk fysiker vid universitetet i Moskva.



Figur 3.5: Kroklinjigt koordinatsystem



Figur 3.6: Skjuvtöjningarna under förutsättning om små vinkeländringar



Figur 3.7: Bestämning av den tangentiella förskjutningen i s-led.

3.2.1 Normalspänning

Förskjutningen i tangentiell led längs med den kroklinjiga s-axeln är

$$v_s = h(s)\varphi(x) \tag{3.13}$$

med $h(s)$ enligt Figur 3.7. Insättning av (3.13) i skjuvtöjningssambandet (3.12) samt av villkoret $\gamma_{xs} = 0$ följt av integrering med avseende på s över ett intervall (s_0, s) ger, med användande av analysens huvudsats

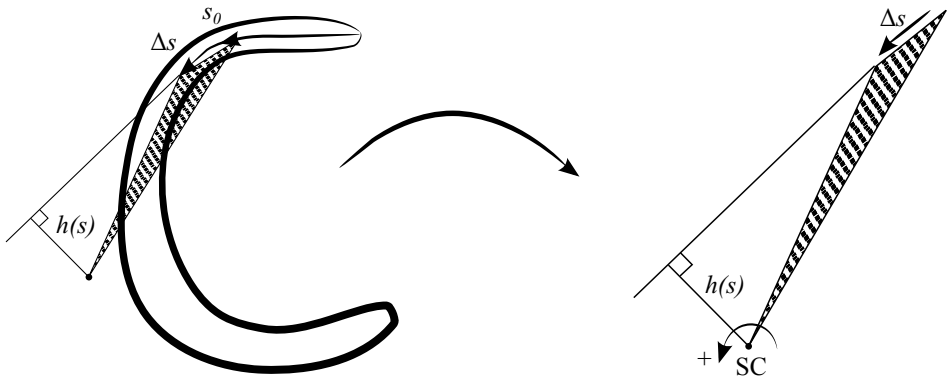
$$u(x,s) = u(x,s_0) - \int_{s_0}^{s} h(s)\,\mathrm{d}s \cdot \frac{\mathrm{d}\varphi(x)}{\mathrm{d}x} \tag{3.14}$$

Definitionen av den så kallade *sektoriella koordinaten* är

$$\Omega(s) = \int_{s_0}^{s} h(s)\,\mathrm{d}s$$

För ett differentiellt inkrement $\mathrm{d}s$ är den sektoriella koordinaten $\Omega(s)$ två gånger beloppet av den triangelyta som ges av $\frac{1}{2}h(s)\mathrm{d}s$, se Figur 3.8. Om $u(x,s_0)$ för enkelhets skull betecknas $u(x)$, ger insättning av den sektoriella koordinaten

$$u(x,s) = u(x) - \Omega(s)\frac{\mathrm{d}\varphi(x)}{\mathrm{d}x} \tag{3.15}$$



Figur 3.8: Geometrisk tolkning av den sektoriella koordinaten för ett differentiellt inkrement ds.

Med användande av Hookes lag och insättning av töjningssambandet $\epsilon_x = \frac{\partial u}{\partial x}$ i (3.15) uttrycks normalspänningen

$$\sigma_x = E\left(\frac{du}{dx} - \Omega(s)\frac{\mathrm{d}^2\varphi}{\mathrm{d}x^2}\right) \tag{3.16}$$

Nu är

$$N(x) = \int_A \sigma_x \,\mathrm{d}A$$

En snittstorhet benämnd *bimomentet* definieras

$$B(x) = \int_A \Omega\,\sigma_x \,\mathrm{d}A$$

Inför vidare benämningarna *sektoriellt moment* S_Ω och *välvtröghetsmoment* I_Ω enligt

$$\begin{aligned} S_\Omega &= \int_A \Omega \,\mathrm{d}A \\ I_\Omega &= \int_A \Omega^2 \,\mathrm{d}A \end{aligned}$$

Då kan snittstorheterna skrivas

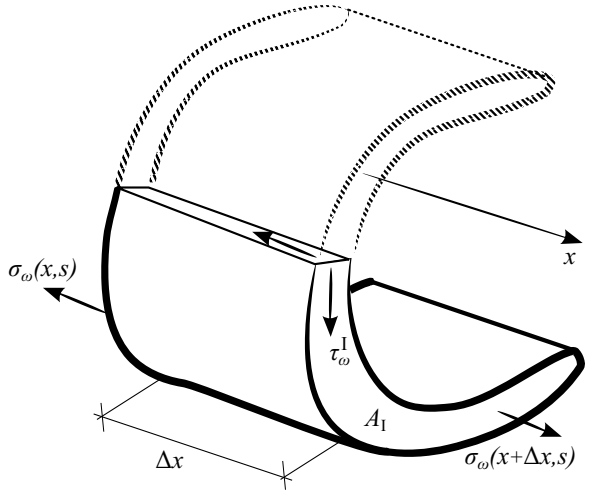
$$\begin{aligned} N(x) &= A\frac{\mathrm{d}u}{\mathrm{d}x} - S_\Omega\frac{\mathrm{d}^2u}{\mathrm{d}x^2} \\ B(x) &= S_\Omega\frac{\mathrm{d}u}{\mathrm{d}x} - I_\Omega\frac{\mathrm{d}^2\varphi}{\mathrm{d}x^2} \end{aligned} \tag{3.17}$$

Om en lämpligt vald konstant subtraheras från den sektoriella koordinaten $\Omega(s)$ fås den så kallade normerade sektoriella koordinaten $\omega(s)$

$$\omega(s) = \Omega(s) - \frac{S_\Omega}{A}$$

Med den normerade sektoriella koordinaten blir snittstorheterna (3.17) okopplade

$$\begin{aligned} N(x) &= EA\frac{\mathrm{d}u}{\mathrm{d}x} \\ B(x) &= -EI_\omega\frac{\mathrm{d}\varphi^2}{\mathrm{d}x^2} \end{aligned} \tag{3.18}$$



Figur 3.9: Axiell jämvikt för en del med snittytan A_{I}.

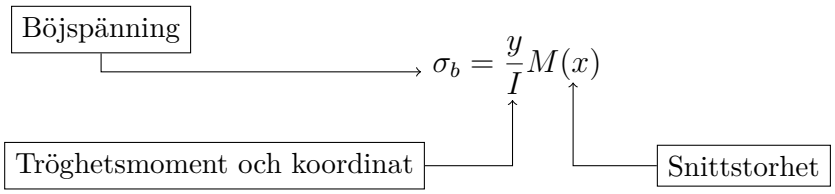
Genom eliminering av förskjutningarna i sambanden för spänning (3.16) och snittkrafter (3.18) fås

$$\sigma_x = \frac{N}{A} + \frac{\omega}{I_\omega} B$$

Det vill säga normalspänningen fås genom att addera effekterna av normalkraft och vridning

$$\sigma_x = \sigma_N + \sigma_\omega$$

Vid ren vridning är $\sigma_N = 0$. Notera likheten med plan böjning



3.2.2 Skjuvspänningar

Materialsambandet $\tau_{xs} = G\gamma_{xs}$ ger att $\tau_{xs} = 0$ eftersom deformationsantagandet säger att $\gamma_{xs} = 0$. Men när normalspänningen σ_x varierar längs med balken måste skjuvspänningar finnas av jämviktsskäl. Axiell jämvikt för en del med snittytan A_{I} ger (se Figur 3.9) under förutsättningen $\sigma_N = 0$ eller σ_N =konstant, att

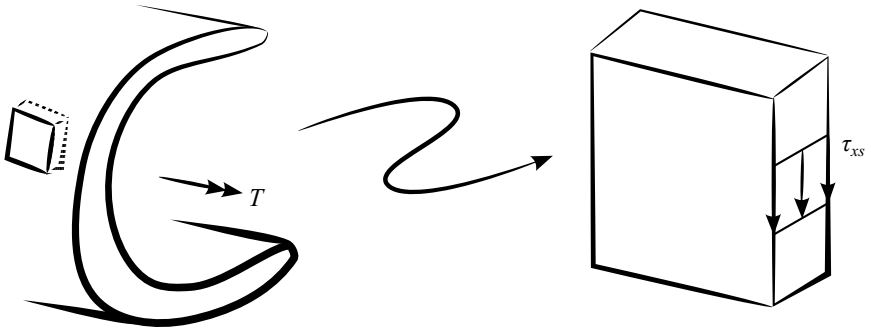
$$\int_{A_{\mathrm{I}}} \sigma_\omega(x + \mathrm{d}x, s)\ \mathrm{d}A - \int_{A_{\mathrm{I}}} \sigma_\omega(x, s)\ \mathrm{d}A - \tau_\omega^{\mathrm{I}} t_{\mathrm{I}}\, \mathrm{d}x = 0$$

vilket medför att

$$\tau_\omega^{\mathrm{I}} = \frac{S_\omega^{\mathrm{I}}}{t_{\mathrm{I}} I_\omega} \frac{\mathrm{d}B}{\mathrm{d}x}$$

Positiv referensriktning för τ_ω^{I} är in i den betraktade delen A_{I}. Notera att skjuvspänningsfördelningen är statiskt ekvivalent med vridmomentet så att

$$T_\omega(x) = \frac{\mathrm{d}B(x)}{\mathrm{d}x} \tag{3.19}$$



Figur 3.10: Skjuvspänningarna är jämnt fördelade över väggtjockleken.

Tabell 3.2: Analogitabell Vlasovsk vridning och böjverkan

Böjverkan	v	q	M	V	I
Vridning	φ	q_ω	$-B$	T_ω	I_ω

Skjuvspänningarna är jämnt fördelade över väggtjockleken, se Figur 3.10. Jämvikt med hänsyn till ett yttre vridmoment q_ω som är fördelat ger

$$\frac{\mathrm{d}T_\omega}{\mathrm{d}x} + q_\omega = 0 \tag{3.20}$$

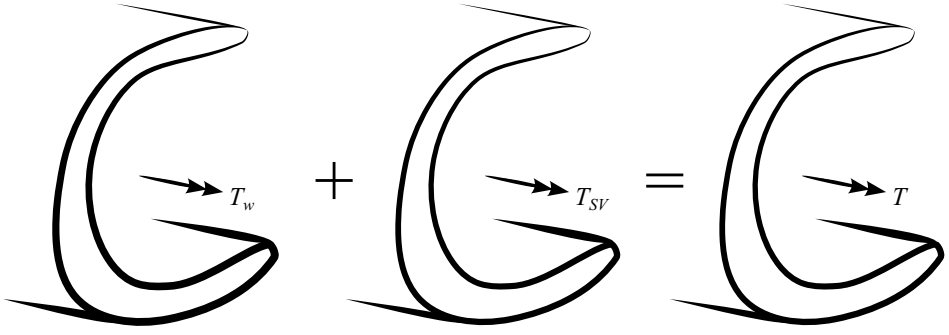
Grundekvationen för ren Vlasovsk vridning ges nu av att sambanden (3.18), (3.19) och (3.20) kombineras genom stegvis insättning

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}\left(EI_\omega \frac{\mathrm{d}^2\varphi}{\mathrm{d}x^2}\right) = q_\omega \tag{3.21}$$

Lösning till (3.21) söks med angivande av de statiska och kinematiska randvillkoren. Eftersom (3.21) till sin form är analog med grundekvationen för plan böjning kan ett Vlasovskt torsionselement i FEM fås genom direkt användning av styvhetsmatrisen för plan böjning. De analoga storheterna anges i Tabell 3.2. [11, 5]

3.3 Blandad vridning

S:t Venantsk och Vlasovsk vridning representerar två separata spännings- och deformationssystem som vart och ett uppfyller jämviktsekvationerna. De kan



Figur 3.11: Skjuvspänningar och resultanter fås genom superposition.

Tabell 3.3: Analogitabell blandad vridning och plan böjning enligt 2:a ordningens teori.

Böjbelastad balk	v	q	I	N	M	Q_y	V	$Q_y - V$
Blandad vridning	φ	q_ω	I_ω	GK_v	$-B$	T	T_ω	T_{SV}

därför superpositioneras. Vinkeln φ kopplar ihop systemen. Vridmomentet vid blandad vridning blir därför

$$T = T_\omega + T_{SV} \tag{3.22}$$

där

$$\begin{aligned} T_\omega(x) = &\quad -EI_\omega \tfrac{\mathrm{d}^3\varphi}{\mathrm{d}x^3} \quad : \text{Vlasovskt vridmoment} \\ T_{SV}(x) = &\quad GK_v \tfrac{\mathrm{d}\varphi}{\mathrm{d}x} \quad : \text{St Venantskt vridmoment} \end{aligned} \tag{3.23}$$

Ekvationerna (3.22) och (3.23) ger tillsammans

$$T = -EI_\omega \frac{\mathrm{d}^3\varphi}{\mathrm{d}x^3} + GK_v \frac{\mathrm{d}\varphi}{\mathrm{d}x} \tag{3.24}$$

Om ett fördelat vridmoment q_ω läggs på balken ger vridjämvikt runt x-axeln

$$\frac{\mathrm{d}T}{\mathrm{d}x} + q_\omega(x) = 0 \tag{3.25}$$

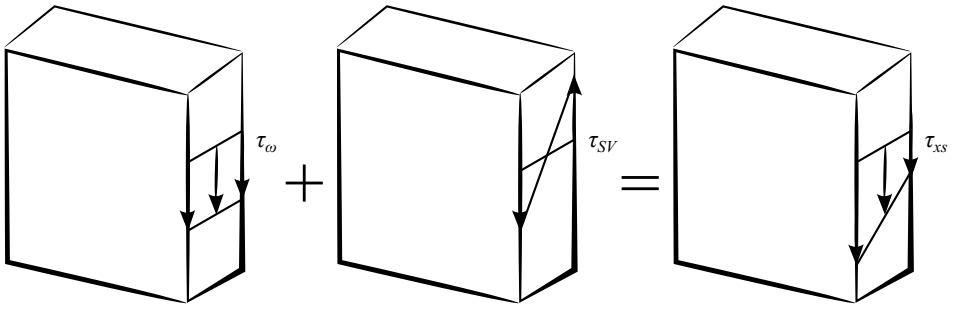
Grundekvationen för blandad vridning fås genom insättning av (3.24) i (3.25)

$$EI_\omega \frac{\mathrm{d}^3\varphi}{\mathrm{d}x^3} - GK_v \frac{\mathrm{d}\varphi}{\mathrm{d}x} = q_\omega \tag{3.26}$$

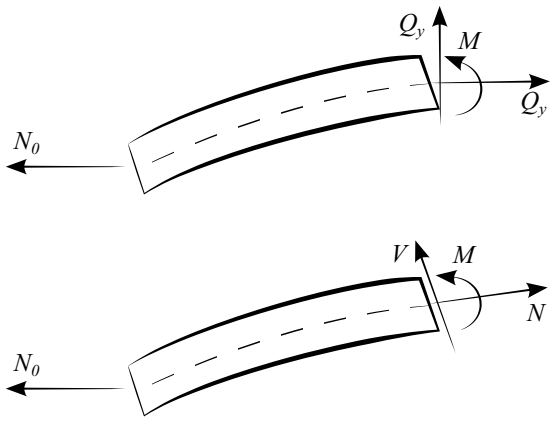
Lösning till (3.26) söks med angivande av de statiska och kinematiska randvillkoren. Den totala skjuvspänningen vid blandad vridning fås genom superposition och illustreras i Figur 3.12. Eftersom (3.26) till sin form är analog med grundekvationen för plan böjning enligt 2:a ordningens teori, kan ett blandat torsionselement i FEM fås genom direkt användning av styvhetsmatrisen för en axiellt belastad balk. De analoga storheterna anges i Tabell 3.3 [11, 5]. Storheten Q_y som anges i Tabell 3.3 ges av

$$Q_y = -EI \frac{\mathrm{d}^3 v}{\mathrm{d}x^3} + N_0 \frac{\mathrm{d}v}{\mathrm{d}x}$$

där v är (plana) nedböjningen transversellt längsriktningen (x-axeln) för en balk med axialbelastningen N_0. Q_y illustreras också i Figur 3.13.



Figur 3.12: Total skjuvspänning vid blandad vridning.



Figur 3.13: Övre delfiguren anger snittstorheterna för ett axialbelastat balkelement i ursprungliga riktningar. Nedre delfiguren anger snittstorheterna i utböjda riktningar.

Figur 3.14: LEGO-modell av F-14 Tomcat från filmen *Top Gun* från 1986.

3.4 Finita elementmetoden

3.4.1 Inledning

Finita elementmetoden (FEM) är en numerisk metod för lösning av partiella differentialekvationer. Historiskt sett har metoden sina rötter i 1940-talets flygindustri med stora beräkningsavdelningar som sysslade med matrisalgebra. Benämningen FEM började dock inte tas i bruk förrän på 1950-talet i USA. Metoden har senare fått en rigorös matematisk grund framförallt i och med Strang & Fix:s *An analysis of the finite element method* från 1973. Terminologin är färgad av de tidiga tillämpningarna på elasticitets- och strukturanalysproblem: *styvhetsmatris*, *lastvektor*, etc. Idag används FEM i många andra tillämpningsområden såsom elektricitet, flödesanalys, akustik och krypning. FEM har den fördelen att komplexa geometrier kan modelleras samt att vissa delar av problemområdet kan ges större noggrannhet än andra. Väderprognoser bygger ofta på en större precision över land än över hav till exempel. Estetiskt sett kan FEM jämföras med ett LEGO-bygge, se Figur 3.14; komplexa strukturer skapas genom (noggrann) sammansättning av enkla element. [22, 25]

3.4.2 Översiktlig metodbeskrivning

Problemområdet diskretiseras i finita element vilka förbinds med varandra i noder (eller knutar som det ibland kallas). Mängden av element och noder kallas nät (eller mesh). En approximativ lösning u till den partiella differentialekvationen söks genom interpolering över de finita elementen och minimering av en residual. Till varje nodpunkt i införs en basfunktion N_i med stöd i nodpunkterna. Basfunktionerna brukar bestå av polynom med låga gradtal och med krav på kontinuitet över elementgränserna. Lösningen u fås som en linjärkombination av basfunktionerna N_i. För att göra matematiken smidigare multipliceras differentialekvationen med en funktion som uppfyller randvillkoren och sedan skrivs differentialekvationen om på stark och svag form. Dessa begrepp härrör från den matematiska grenen variationskalkyl som behandlar optimering av

funktionaler. För att öka precisionen i modellen kan antingen basfunktionernas gradtal ökas eller nätets indelning i finita element förfinas. Sammankopplingen av nät och basfunktioner kallas assemblering. Nät, laster och randvillkor benämns FE-modell. [22, 25] Avsnittet som följer är en sammanfattning av standardmetoden baserad på framställningen i [4] och [25].

3.4.3 Standardmetoden

Betrakta en kropp med volymen V, densiteten ρ, ytan S, ytlasten $\mathbf{t}$ och volymlasten $\mathbf{b}$ såsom illustreras i Figur 3.15. Om $\mathbf{u}$ är förskjutningsvektorn så kan Newtons andra lag uppställas [4]

$$\int_S \mathbf{t}\ \mathrm{d}S + \int_V \mathbf{b}\ \mathrm{d}V = \int_V \rho\ddot{\mathbf{u}}\ \mathrm{d}V \tag{3.27}$$

Statisk kraftjämvikt innebär att högerledet i (3.27) är lika med nollvektorn:

$$\int_S \mathbf{t}\ \mathrm{d}S + \int_V \mathbf{b}\ \mathrm{d}V = \mathbf{0} \tag{3.28}$$

Inför ett rätvinkligt koordinatsystem. Då är:

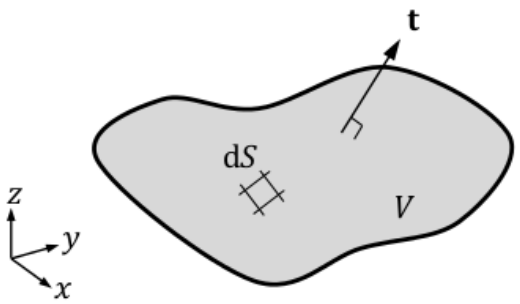
$$\mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}$$

Med hjälp av Gauss divergenssats kan jämviktsekvationen (3.28) omskrivas [4]:

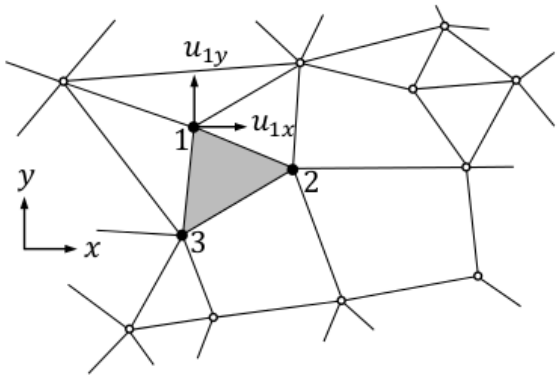
$$\int_V (\tilde{\nabla}^T \boldsymbol{\sigma} + \mathbf{b})\ \mathrm{d}V = \mathbf{0} \tag{3.29}$$

Här betecknar $\boldsymbol{\sigma}$ spänningsvektorn och $\tilde{\nabla}$ en differentialoperator enligt

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{bmatrix}, \qquad \tilde{\nabla} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \end{bmatrix}$$



Figur 3.15: Frilagd kropp med volymen V, densiteten ρ, ytan S, snittkraften $\mathbf{t}$ och inre kraften $\mathbf{b}$



Figur 3.16: Illustration över ett finita elementnät med noder

Eftersom kroppens volym från början var godtycklig representerar (3.29) den starka formen av jämviktsekvationen:

$$\tilde{\nabla}^T \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \tag{3.30}$$

Idén i det följande är att uttrycka $\boldsymbol{\sigma}$ i förskjutningarna $\mathbf{u}$ med hjälp av ett konstitutivt samband (till exempel Hookes lag) och sedan approximera $\mathbf{u}$ på lämpligt vis. Emellertid inses att denna approximation måste vara åtminstone en gång differentierbar. Kravet sänks genom att skriva om jämviktsekvationen ytterligare. För effektiviteten i de numeriska beräkningarna är detta en fördel. Multiplicera den starka formen (3.30) med en godtycklig vektor $\mathbf{v}$, integrera över volymen V och tillämpa Gauss divergenssats en gång till för att erhålla den svaga formen av jämviktsekvationen

$$\int_V (\tilde{\nabla}\mathbf{v})^T \boldsymbol{\sigma} \, \mathrm{d}V = \int_S \mathbf{v}^T \mathbf{t} \, \mathrm{d}S + \int_V \mathbf{v}^T \mathbf{b} \, \mathrm{d}V, \quad \mathbf{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{3.31}$$

Inför ett elementnät med noder i princip enligt Figur 6.3. Om nodförskjutningarna för ett element $\mathbf{a}^e$ är kända kan också förskjutningarna mellan noderna i elementet uppskattas genom interpolation, jämför Figur 3.17. Med hjälp av basfunktioner $\mathbf{N}^e$ kan detta uttryckas som [25]:

$$\mathbf{u}^e \approx \mathbf{N}^e \mathbf{a}^e$$

För konvergens fordras att basfunktionerna $\mathbf{N}^e$ är åtminstone en gång differentierbara och att de varierar kontinuerligt över elementgränserna. För sådana basfunktioner $\mathbf{N}^e$ kan förskjutningarna beskrivas med en differentierbar funktion med stöd i elementnoderna och det gäller approximativt att

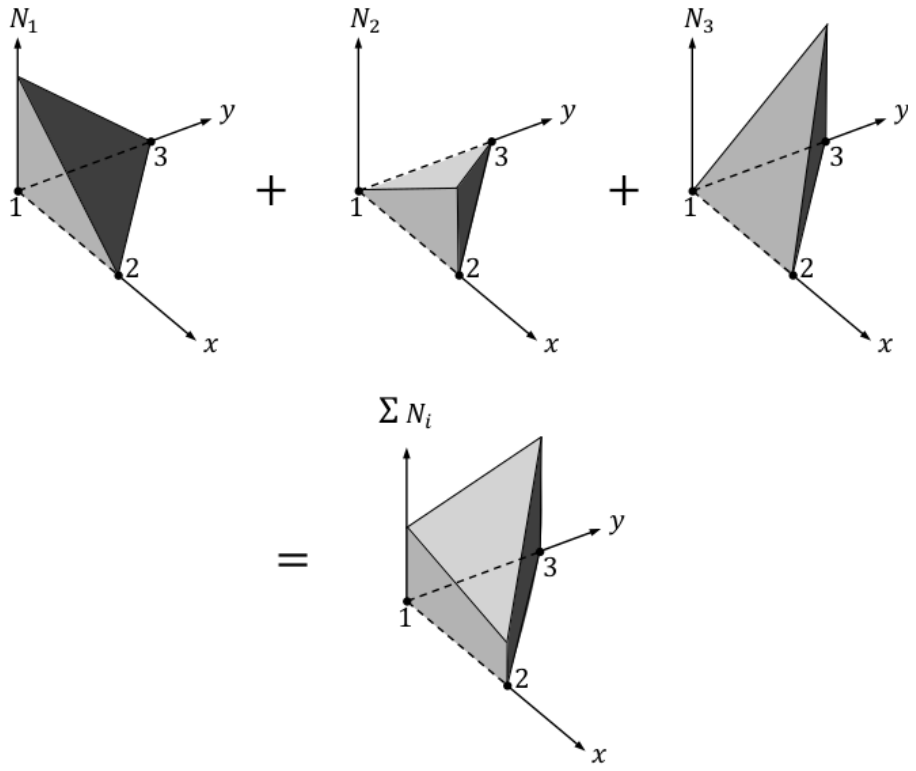
$$\mathbf{u} \approx \mathbf{N}\mathbf{a}$$

Om $\mathbf{B} = \tilde{\nabla}\mathbf{N}$ fås töjningarna

$$\boldsymbol{\epsilon} = \mathbf{B}\mathbf{a} \tag{3.32}$$

Enligt Galerkins metod ansätts [25]

$$\mathbf{v} = \mathbf{N}\mathbf{c}$$



Figur 3.17: Med hjälp av basfunktioner kan variationen inom ett finita element uppskattas genom interpolation av nodvärdena.

där det faktum att $\mathbf{v}$ är en godtycklig vektor medför att $\mathbf{c}$ är detsamma. Insättning i den svaga formen (3.31) ger

$$\int_V \mathbf{c}^T \mathbf{B}^T \boldsymbol{\sigma} \, \mathrm{d}V = \int_S \mathbf{c}^T \mathbf{N}^T \mathbf{t} \, \mathrm{d}S + \int_V \mathbf{c}^T \mathbf{N}^T \mathbf{b} \, \mathrm{d}V$$

där vektorn $\mathbf{c}$ kan elimineras eftersom den är godtyckligt vald och oberoende av läget i kroppen. Då fås:

$$\int_V \mathbf{B}^T \boldsymbol{\sigma} \, \mathrm{d}V = \int_S \mathbf{N}^T \mathbf{t} \, \mathrm{d}S + \int_V \mathbf{N}^T \mathbf{b} \, \mathrm{d}V \tag{3.33}$$

Observera att de ursprungliga jämviktsekvationerna är giltiga för varje konstitutivt samband mellan spänning $\boldsymbol{\sigma}$ och töjning $\boldsymbol{\epsilon}$. Inför nu ett specifikt konstitutivt samband. Med Hookes lag fås

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\epsilon}$$

vilket med hjälp av approximationen i (3.32) ger

$$\boldsymbol{\sigma} = \mathbf{DBa} \tag{3.34}$$

Insättning av (3.34) i (3.33) ger finita elementformuleringen

$$\int_V \mathbf{B}^T \mathbf{DB} \, \mathrm{d}V \, \mathbf{a} = \int_S \mathbf{N}^T \mathbf{t} \, \mathrm{d}S + \int_V \mathbf{N}^T \mathbf{b} \, \mathrm{d}V \tag{3.35}$$

vilken förkortat omskrivs

$$\mathbf{Ka} = \mathbf{f}$$

med

$$\mathbf{K} = \int_V \mathbf{B}^T \mathbf{D} \mathbf{B} \, \mathrm{d}V$$

och

$$\mathbf{f} = \int_S \mathbf{N}^T \mathbf{t} \, \mathrm{d}S + \int_V \mathbf{N}^T \mathbf{b} \, \mathrm{d}V$$

Lösningen av jämviktsekvationerna förutsätter kännedom om randvillkoren. För varje del av kroppens yta S måste antingen ytlasten $\mathbf{t}$ eller förskjutningen $\mathbf{u}$ vara känd, de väsentliga och naturliga randvillkoren [25].

4 Beräkningsmiljön

Beräkningsmiljön består av Python och de fyra paketen PyQt, Visvis, PyCALFEM och NumPy.

4.1 Python

Python är ett så kallat ”general-purpose” objektorienterat programmeringsspråk som lanserades 1989 av Holländaren Guido van Rossum [30]. Namnet inspirerades van Rossum till då han läste manuskriptet till Monty Python's Flying Circus, en BBC-serie från 1970-talet [10]. Med objekt avses inom datavetenskapen en klass som innehåller datafält och metoder [14]. Enligt TIOBE Programming Community Index positionerar sig Python bland de tio populäraste programmeringsspråken år 2013 [39]. Python har använts av bland andra Google, Yahoo!, CERN och NASA och ingår som en standardkomponent i flera operativsystem, bland andra Linux och OS X [9, 2, 32]. Python positionerar sig som ett läsbart och användarvänligt språk, vilket kommer till uttryck i 1) den så kallade off-side-regeln[1] som innebär att deklarerade block skall indenteras. 2) Kodorden liknar dem i C och Java, men oftare används i Python engelska ord, exempelvis `and`, `or` och `not` i stället för `&&`,`||` och `!`. 3) Metoder anropas explicit med `instance.method(argument)` i stället för `Class.method(instance, argument)` såsom i C++ och Java; en syntaktisk omskrivning vars syfte är att öka läsbarheten. Alex Martelli [19] sammanfattar ambitionen i Python med orden: ”To describe something as clever is not considered a compliment in the Python culture.”

4.2 PyQt

Qt är ett GUI-utvecklingsverktyg. GUI är en förkortning av Graphical User Interface och betecknar användargränssnittet. PyQt är en Python-portering av Qt. Mjukvaran är gratis och utvecklad av det brittiska företaget Riverbank Computing. Det är plattformsoberoende, det vill säga det stöder såväl

[1] Off-side-regeln definierades så här av Peter J. Landin år 1966: ”Any non-whitespace token to the left of the first such token on the previous line is taken to be the start of a new declaration.” [17]

Windows, OS X som Linux, och omfattar 620 klasser och över 6000 metoder, däribland

- så kallade GUI widgets (mer om det nedan)
- SQL-databashantering (ODBC, MySQL, PostgreSQL, Oracle)
- XML-parser

En widget är ett objekt i GUI:n, t ex en knapp, en lista, en combo-box eller en textrad. PyQt består av bland andra följande moduler

- QtCore. Innehåller icke-GUI-funktioner såsom eventloopar, signal och kontaktmekanismer och användarinställningar.
- QtGUI. Innehåller de flesta GUI-funktioner (widgets med mera).
- QtNetwork
- QtOpenGl

Adobe Photoshop Album och Google Earth är två exempel på gränssnitt som utvecklats med Qt. Karaktäristiskt för Qt är eventmekaniken med signaler och kontakter. Signaler kan vidarebefordra data i ett typsäkert läge, det vill säga så kallade *callback*-fel som var vanliga i äldre system undviks. [27, 28]

4.3 Visvis

Visvis är ett Python-bibliotek för objektorienterad visualisering i 1D-4D som bygger på OpenGL-teknik. Visvis styrka ligger i 3D-4D-visualisering och snabb rendering med OpenGl jämfört med bibliotek som Matplotlib som endast hanterar 1D-2D-visualisering. Den som arbetat i Matlab känner igen sig i anrop såsom `plot()` och `surf()`. Beräkningsmiljön bygger på så kallade widget objects eller *wibjects* och world objects eller *wobjects*. Dessa bygger upp en trädliknande struktur tillsammans med huvudfönstret där wibjects omfattar sådant som koordinatfönster med en kameravinkel (den så kallade `Axes`-klassen). Wobjects omfattar linjer, bilder, 3D-volymer och ytnät. Dessa utgör grenarna i trädstrukturen. Det är enkelt att manipulera wibjects och wobjects utseende genom att ändra på deras egenskaper. [42]

4.4 PyCALFEM

CALFEM är en verktygslåda med MATLAB-rutiner för finita elementmodellering och är förkortning för Computer aided learning of the finite element method. Det uttalade syftet är att vara ett pedagogiskt instrument för inlärning av FEM. CALFEM började utvecklas i slutet av 70-talet och i den nuvarande versionen (3.4) finns rutiner för såväl fjäderelement, stång- och balkelement som skiv- och solidelement. I flera fall finns elementformuleringar för geometrisk icke-linjäritet. Det finns också rutiner för dynamiska system och visualisering. PyCALFEM är helt enkelt en portering till Python och omfattar i dagsläget

flertalet rutiner. Emellertid saknas till exempel rutiner för skapande och assemblering av solidelement. Dessa rutiner har i förekommande fall porterats av författaren och finns angivna med källkod i Bilaga 3.

4.5 NumPy

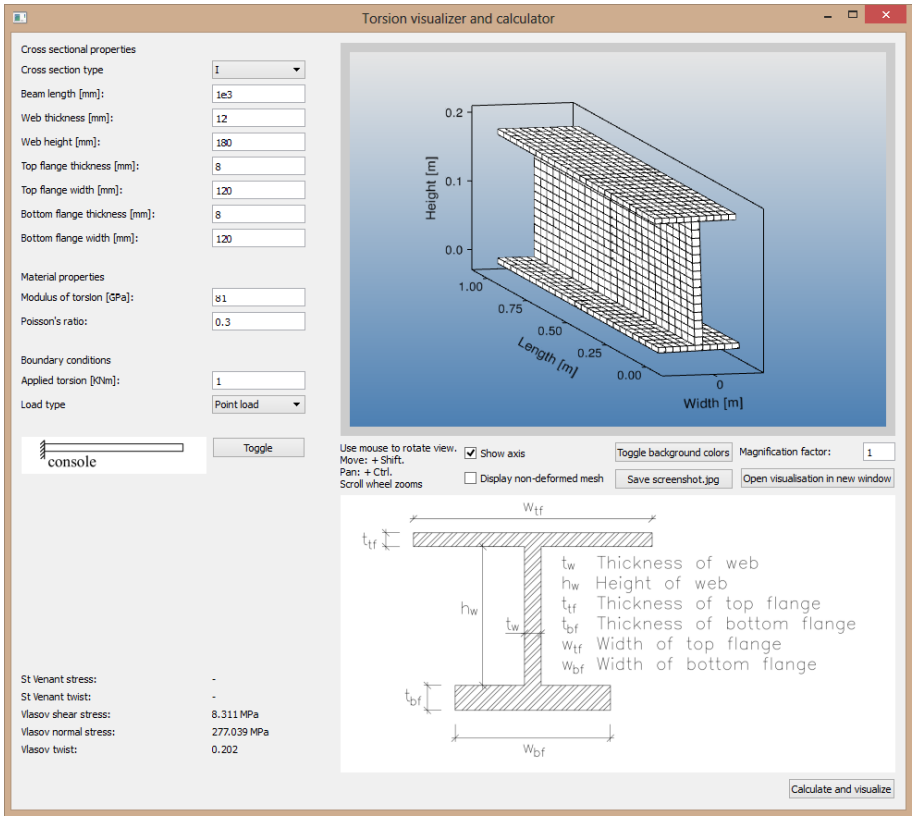
NumPy är ett tillägg till Python med stöd för stora, flerdimensionella matriser tillsammans med ett bibliotek med matematiska funktioner. NumPy är öppen källkod, implementerat i C och kan använda flera beräkningskärnor. Styrkan ligger i den snabbhet och effektivitet som kan uppnås i beräkningen när algoritmerna uttrycks i huvudsak med matriser och tillhörande operatorer, även om Python är ett högnivåspråk. NumPy är likt MATLAB i funktion och har sina rötter i Numeric – ett matrisbibliotek för Python från mitten av 90-talet. Det dök upp ett antal olika sorters matrisbibliotek under de efterföljande åren. Travis Oliphants strävan efter ett förenat matrisbibliotek för Python resulterade år 2006 i den första versionen av NumPy. [24]

5 Programmet

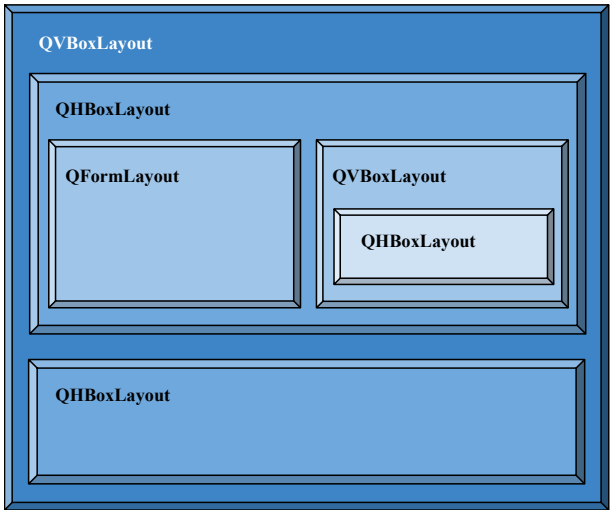
5.1 Gränssnitt

Gränssnittet omfattar ett fönster där vänstra halvan består av indatafält och högra halvan består av visualisering och grafik, se Figur 5.1. Längst ner i högra hörnet finns en beräkningsknapp vars funktion är lik "Enter"-knappen på en miniräknare. Indatafältet är indelat i avsnitt för geometri, material och randvillkor. Målet har varit att bereda användaren en enkel kontrollpanel för beräkning och visualisering i 3D av balkvridning.

Användargränssnittet är implementerat i Qt. Huvudfönstret är indelat i ett antal (så kallade) *layouts*. Organisationen framgår av Figur 5.2 där de huvudsakliga layouts:en finns illustrerade. Med figurens beteckningar utgör en `QVBoxLayout` stommen. En `QVBoxLayout` är en vertikalt organiserad behållare som kan fyllas med widgets eller nya layouts. I `QVBoxLayout`:en ligger direkt två stycken `QHBoxLayouts` som är behållare med horisontell organisation, det vill säga som fylls med andra layouts eller widgets i horisontell ordning. Den undre `QHBoxLayout`:en innehåller en knapp som är förskjuten till nedre högra hörnet med hjälp av ett `addStretch`-kommando. Den övre `QHBoxLayout`:en innehåller ett formulär till vänster och en ny `QVBoxLayout` tll höger med två figurer. Mitt emellan figurerna finns en `QHBoxLayout` med ett antal kontrollverktyg för den grafiska presentationen av balken i 3D. I `QFormLayout:en` kan användaren mata in tvärsnittsdata, materialdata och definiera randvillkoren. En `QFormLayout` är som ett formulär. Varje rad som fogas till formuläret består av två delar, typiskt en textremsa och ett indatafält eller liknande. Längst upp i formuläret väljer användaren med hjälp av en rullgardinsmeny en profil (antingen I, U eller Z). När användaren väljer en profil uppdateras ena figuren till höger om formuläret. Figuren föreställer tvärsnittet med information om hur tvärsnittsmåtten ska tolkas (t ex flänstjocklek, livtjocklek med mera). Längst ner i formuläret anges, efter att beräkningsknappen tryckts och beräkningsalgoritmerna genomlöpts, de aktuella spänningarna och rotationerna enligt S:t Venant och Vlasov. När användaren klickar på beräkningsknappen i nedre högra hörnet ritas också en 3D FE-modell av balken upp.



Figur 5.1: Gränssnittet består av ett fönster med indatafält och grafik.



Figur 5.2: Användargränssnittet består av ett huvudfönster indelat i ett antal *layouts* varav de huvudsakliga här finns illustrerade.

Tabell 5.1: Klasser med förklaring

Klass	Förklaring
Window	Huvudfönstret innehåller de flesta widgets (indatafält, textfält, listor, checkboxar, knappar med mera). Det finns bland annat en metod som kontrollerar att indata består av giltiga teckenkombinationer och en metod som låser flänsbredderna och tjocklekarna till varandra i fallet med U- och Z-tvärsnitt
Beam	All data om balken (längd, tvärsnittsmått etc) innehålls i Beam-klassen
St_Venant	De S:t Venantska spänningarna och rotationerna lagras i en egen klass
Vlasov	De Vlasovska spänningarna och rotationerna lagras i en egen klass
FEM	Informationen från FE-modelleringen av balken för 3D-visualisering (nodförskjutningar, frihetsgrader med mera) lagras i en egen klass
WorkThread	När klassobjektet startas skickas signaler i tur och ordning vilka påbörjar var sin tråd

5.2 Uppbyggnad av koden

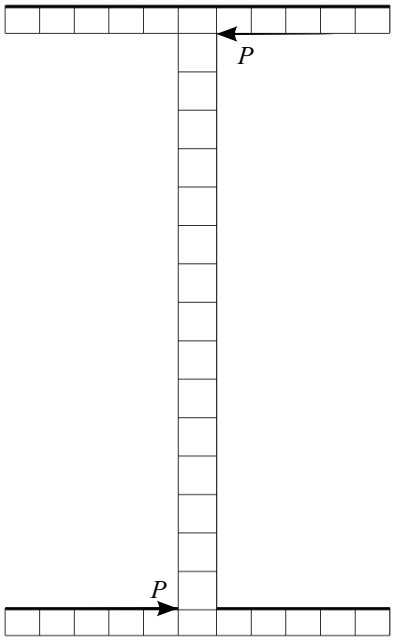
Pythonkoden är uppbyggd av 7 huvudklasser. Klasserna finns angivna i Tabell 5.1 tillsammans med förklaringar. Fönsterklassen `Window` innehåller samtliga `layouts` och `widgets`. Huvudfönstrets indelning i `layouts` framgår av Figur 5.2. Se också Bilaga 2 som innehåller den kommenterade källkoden.

Hur det finita 3D-punktnätet har numrerats framgår av figurerna i Bilaga 1. När det gäller numreringen av linjestyckena som sammanbinder noderna hänvisas till källkoden med användande av följande kommando (som uppritar geometrin med linjenumrena). Det är det enklaste sättet att varsebli numreringsmönstret. Geometrin tillsammans med numreringar kan då vridas och vändas i 3D och synliggöras.

```
pcv.drawGeometry(g, axes=None, axesAdjust=True,
    drawPoints=True, labelPoints=False, labelCurves=True,
    title=None, fontSize=14, N=20)
```

För övrigt ingår även en modul med följande tre delar

- En portering av CALFEM-rutinen `soli8e` som skapar elementmatisen för ett 8-nods isoparametriskt solidelement



Figur 5.3: Kraftparet P är statiskt ekvivalent med vridmomentet.

- En anpassning av PyCALFEM-rutinen `applyforce` som även stöder 3D-element
- En anpassning av PyCALFEM-rutinen `applybc` som även stöder 3D-element

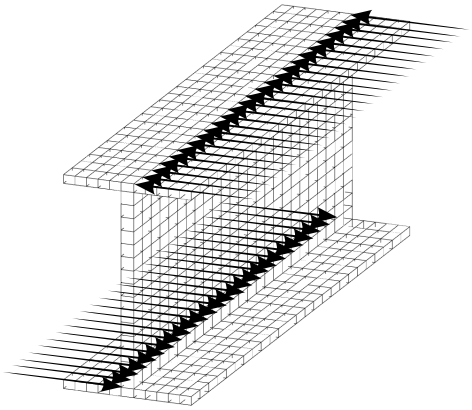
Modulen återfinns i Bilaga 3.

5.3 FE-modell

Finita elementmodellen består av isoparametriska 8-nods solidelement. Det totala antalet element är 925. Balklivet består av 25 gånger 15 element och varje fläns består av 25 gånger 11 element. Antalet element är optimerat för att användas på en dator med 4 GB RAM.

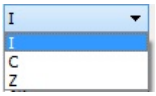
Det statiska randvillkoret åstadkoms genom ett med vridmomentet statiskt ekvivalent kraftpar som angriper i livets topp och botten, se Figur 5.3 för en illustration. Det fördelade vridmomentet är statiskt ekvivalent med en mängd kraftpar, lika många som antalet noder i x-led. Se Figur 5.4 för en illustration över dessa kraftpars angreppspunkter.

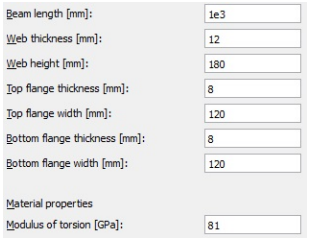
De kinematiska randvillkoren åstadkoms i fallet med konsolen genom att samtliga noder i ena balkänden låses i tre frihetsgrader (translation). I fallet med gaffellagring låses samtliga noder i ena balkänden i y-led. En nod i livet låses i x- och z-led. På grund av symmetrivillkoret låses samtliga noder i andra balkänden i y-led. I fallet med fast inspänning låses samtliga noder i ena balkänden i tre frihetsgrader (translation). På grund av symmetrivillkoret låses samtliga noder i andra balkänden i y-led.

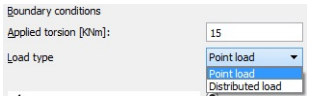


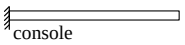
Figur 5.4: Ett fördelat vridmoment är statiskt ekvivalent med en mängd kraftpar, lika många som antalet noder i x-led.

5.4 Användarhandledning: Enkelsymmetriskt I-tvärsnitt

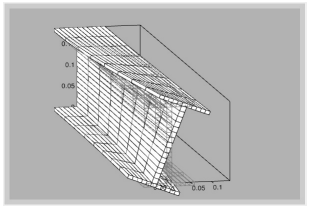
1. Användaren väljer I-tvärsnittet i rullgardinsmenyn. 

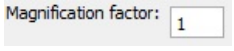
2. Tvärsnittets form och materialegenskaper specificeras.


3. Vridmomentet eller ett fördelat vridmoment anges.


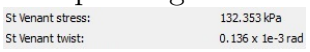
4. Upplagsvillkoren anges med en toggle-knapp.


5. Användaren klickar på knappen för att beräkna och visualisera.


6. Balken visualiseras i 3D. Användaren kan vrida och vända, flytta och panorera med hjälp av musen, shift- och ctrl-tangenterna. Det går att zooma med musens scroll-hjul.


7. Deformationerna kan ändras med en förstoringsskala.


8. Balken kan visualiseras med eller utan en bakgrundsbild av det odeformerade finita elementnätet. 

9. I nedre vänstra hörnet visas de aktuella spänningarna och rotationerna enligt St Venants och Vlasovs teori. 

6 Avslutning

6.1 Slutsatser

Att integrera dynamisk programvara i undervisningen kan vara ett sätt att tillämpa en sociokulturell syn på kunskap, det vill säga kunskap som resultatet av en kommunikativ process, och välanpassad programvara kan tjäna som en stödstruktur för eleven i den närmaste utvecklingszonen. Att integrera programvara som ett laborativt och pedagogiskt instrument i undervisningen kan vara ett steg i riktning mot att göra datorn till en artefakt snarare än ett maskulint statusobjekt. Ett användarvänligt gränssnitt baserat på öppen källkod kan med fördel utvecklas för att visualisera vridna balkar i 3D med hjälp av FEM och för att räkna ut spänningar och rotationer enligt S:t Venants och Vlasovs teori.

S:t Venantsk vridning bygger på antagande om noll normalspänningar svarande mot fri välvningsdeformation av balktvärsnitt. Skjuvspänningarna kan bestämmas med såphinneanalogin och vridstyvhetens tvärsnittfaktor. Rotation och spänningar kan approximeras med FEM genom analogin med stångverkan. Vlasovsk vridning bygger på antagandet att skjuvtöjningarna försummas. Normalspänningen fås som en funktion av välvstyvhetens tvärsnittsfaktor och den normerade sektoriella koordinaten. Skjuvspänningarna fås som en funktion av välvstyvhetens tvärsnittsfaktor och det statiskt sektoriella momentet. Rotation och spänningar kan approximeras med FEM genom analogin med plan balkböjning. S:t Venantsk och Vlasovsk vridning representerar två separata spännings- och deformationssystem som vart och ett uppfyller jämviktsekvationerna. Blandad vridning fås genom superposition där vinkeländringen kopplar ihop systemen. Rotation och spänningar kan beräknas genom analogin med plan balkböjning enligt andra ordningens teori.

Python är ett användarvänligt, tydligt och objektorienterat programmeringsspråk. Till Python finns ett flertal paket och moduler med vilkas hjälp FE-beräkningen, visualiseringen och det grafiska gränssnittet kan genomföras och implementeras. NumPy erbjuder effektiv matrishantering i flera dimensioner så länge skalärer och matriser inte blandas för mycket. Den som någon gång skrivit kod i MATLAB kommer lätt att känna igen sig. PyCALFEM drar nytta av NumPy:s matrisfunktioner för att göra FE-beräkningar. Med Visvis visualiseras det deformerade 3D-elementnätet och renderas i ett fönster där användaren kan vrida och vända, zooma in och ut och flytta vyn. Renderingen görs med

OpenGL-teknik och är därför snabb. Användargränssnittet implementeras med PyQt vilket medför att det är plattformsoberoende, det vill säga fungerar lika väl i Windows som i OS X och Linux. Användaren kan så att säga ställa frågor till programmet: "Vad händer om man ändrar den här parametern...". Rotationen och välvningen som funktion av tvärsnittets geometri kan undersökas interaktivt liksom effekten av upplags- och lastvillkoren.

Bland svagheterna i det utvecklade programmet hör beräkningshastigheten. Det tar tid att genomföra FE-beräkningarna; allt mellan en sekund och några sekunder. Det hade varit intressant med visualisering i realtid allteftersom användaren ändrar på tvärsnitts- och/eller materialdata samt randvillkor. Detta hade också ökat förståelsen för de visualiserade fenomenen på ett helt annat sätt. Den genomsnittliga beräkningstiden innebär emellertid att uppdateringar i realtid är olämpliga och istället begagnas en knapp i nedre högra fönstret som fungerar som "Enter"-symbolen på en miniräknare. En annan svaghet är begränsningen i antal noder och finita element. Det går inte att göra det finita elementet särskilt fint med mindre än att minnet på en genomsnittlig dator tar slut. Det som fattas är att använda sparse-matriser och en annan lösare.

6.2 Diskussion

Balkvridning som fenomen inbegripet välvning kan vara svårbegripligt för en nybörjare. Att göra sig en bild av vad som händer med utgångspunkt i de analytiska uttrycken allena är mer eller mindre komplicerat. Programvaran som utvecklas i den här uppsatsen (se Bilaga 2 för källkoden) ger nybörjaren möjlighet att nalkas fenomenet på ett annorlunda och intuitivt sätt. Användaren har direkt möjlighet att prova sig fram, vrida och vända på ett visualiserat balkelement och så att säga ställa frågor till programmet: "'Vad händer om jag gör så här... ". Det här leder till ett helt annat vis att förstå balkvridning på än genom de analytiska uttrycken. Det ena utesluter naturligtvis inte det andra. En lärare skulle kunna dra nytta av programvaran i sin undervisning genom att göra det till ett laborativt moment i utbildningen. En elev skulle kunna få till uppgift att undersöka spörsmål såsom: "Hur ser ett tvärsnitt ut som ger liten välvning?" Frågan är öppen och möjliggör ett undersökande arbetssätt där samband kan upptäckas. Den öppna frågan kan sedan förankras i de analytiska uttrycken i balkteorin. Då finns potential att se matematik *genom* problemlösning, för att tala med Kerstin Ekstig [13]. Detta är en aspekt av dynamisk programvara [7]. Programmet tjänar som stödstruktur för eleven i den närmaste utvecklingszonen och bildar en bro mellan den punkt där eleven befinner sig för tillfället och den punkt som är målet för studierna [8, 18].

Balkvridning är bara ett i mängden av fysikaliska fenomen som kan visualiseras och begreppsliggöras enligt principerna och metoderna i den här rapporten. Den som förstår grunderna i Python-programmering inbegripet implementeringen av det grafiska gränssnittet i PyQt kan använda PyCALFEM som en verktygslåda för att visualisera och räkna på till exempel elektriska fält, värmeflöden, vätskeflöden, akustik med mera. Andreas Ottosson [26] visar i sitt examensarbete från 2010 hur PyCALFEM kan användas för att till exempel beräkna (det

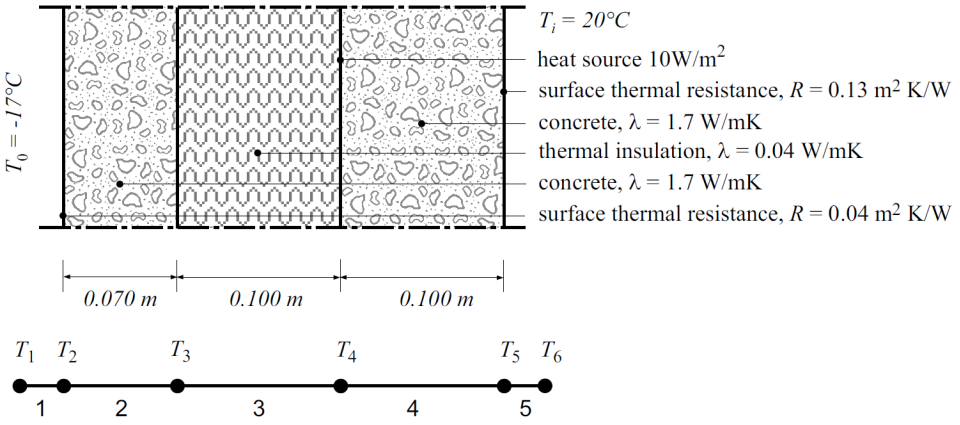
endimensionella) värmeflödet genom en vägg bestående av ett skikt mineralull mitt emellan två skikt av betong, eller nedböjningen av en fritt upplagd balk, se Figur 6.1 och 6.2. Andreas Edholm [6] visar i sitt examensarbete från 2013 hur man kan generera effektiva nät i 1D-3D i PyCALFEM. Ett exempel på ett sådant nät visas i Figur 6.3. Sådana nät kan användas för att modellera exempelvis värmeflöde i 2D.

Utmaningen är att skapa ett användarvänligt gränssnitt som fungerar som ett didaktiskt redskap och att ge programvaran sin rätta plats i undervisningen. Som Klafkis modell med den didaktiska triangeln visar, jämför Kapitel 2, så skall den dynamiska programvaran inte blott vara en del av undervisningens innehåll utan fungera som en kommunikativ partner som stödjer elevens tänkande [34]. En lärare som skräddarsyr och implementerar denna programvara i sin undervisning har goda möjligheter att ge fysik- och framför allt matematikundervisningen de inslag av laboration och undersökande arbetssätt som traditionellt saknas, jämför [7]. Vad som behövs är möjligheter till spekulation, argumentation och verifiering [13]. Då kan programvara tjäna som en länk mellan den punkt där eleven befinner sig just nu och den punkt som är målet för studierna.

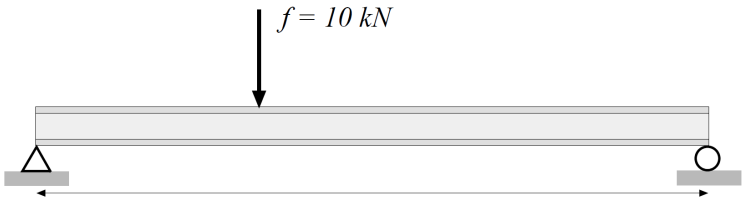
I läroplanen för gymnasieskolan framgår det under examensmålen för Naturvetenskapsprogrammet att "[e]xperiment, laborationer [...] och andra jämförbara praktiska moment ska [...] vara centrala inslag i utbildningen." [36] Detta ställer tydliga krav på inte minst matematiklärare. Matematikundervisningen skall enligt ämnesplanen "innehålla varierade arbetsformer och arbetssätt" inbegripet "undersökande aktiviteter" och eleverna skall "ges möjlighet att utveckla sin förmåga att använda digital teknik" och "digitala medier" [38]. Ämnesplanen i fysik anger att eleven skall ges möjlighet att "använda datorstödd utrustning för [...] simulering, beräkning [...] av data" [37]. I kursen Matematik 5 är "användning och lösning av differentialekvationer med digitala verktyg" ett centralt innehåll [38]. I kursen Fysik 3 anges som ett centralt innehåll användandet av "datorbaserad numerisk simulering [...] för att fördjupa och tillämpa valfritt område på en problemställning med anknytning till fysik" [37]. Det är lätt att se hur metoderna i föreliggande uppsats kan användas för att på olika vis uppfylla styrdokumenten för gymnasieskolan. Balkvridning är en konkret problemställning med anknytning till fysik och en elev skulle kunna bekanta sig med fenomenet genom det grafiska gränssnittet utan att för den skull behöva fördjupa sig i detaljerna i balkteorin om S:t Venantsk och Vlasovsk vridning. Mer generellt kan användarvänliga gränssnitt för simulering av fenomen såsom elektriska fält, vätskeflöden, akustik med mera implementeras i PyQt med hjälp av PyCALFEM enligt metoderna i föreliggande uppsats.

6.3 Sammanfattning

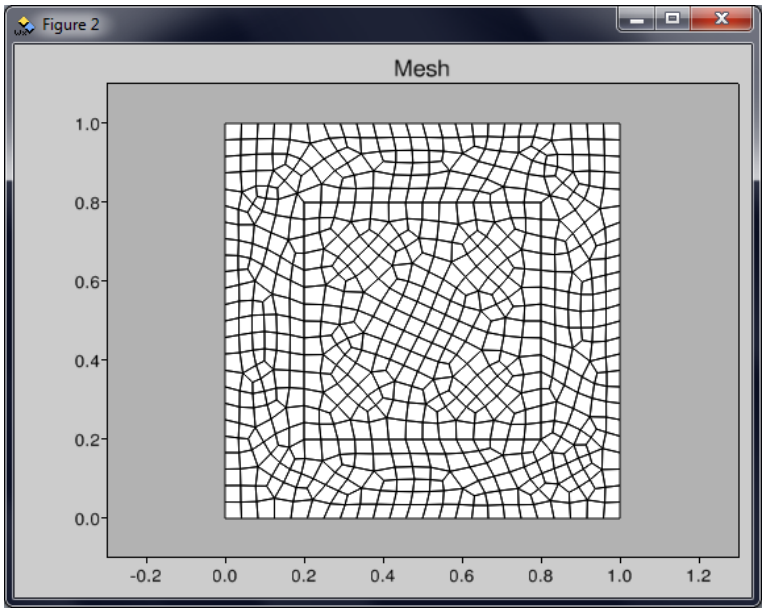
Skolan har ingen lång tradition av att arbeta laborativt i matematik och dynamisk programvara är ett sätt att fylla denna brist. Dynamisk programvara låter eleven gå i dialog, utmanar, skapar nyfikenhet, åskådliggör begrepp och underlättar inlärningen. Det krävs emellertid att lärare är villiga att sätta sig in i de relativt nya möjligheterna eftersom datoranvändning i matematik- och fy-



Figur 6.1: Vägg i genomskärning för beräkning av värmeledning. Hämtad från [1].



Figur 6.2: Fritt upplagd balk. Hämtad från [1].



Figur 6.3: Finita elementnät genererat i PyCALFEM. Hämtad från [6].

sikundervisningen inte per automatik genererar goda resultat. Den som är villig att investera lite tid i Python-programmering kommer med relativt enkla medel att kunna skräddarsy programvara åt sina elever. Ett användarvänligt program för visualisering i 3D av en vriden balk och för beräkning av maximala spänningar och rotationer i tvärsnittet utvecklas i programmeringsspråket Python. Programmet bygger på en rad paket och moduler baserade på öppen källkod: Visvis, NumPy, PyCALFEM och PyQt. Den visualiserade balkkroppen kan vridas och vändas, flyttas och panoreras med enkla musklick och tangenttryck. 3D-figurens deformationer beräknas med FEM. Tre sorters öppna, tunnväggiga tvärsnittsprofiler är tillgängliga: enkelsymmetriska I-tvärsnitt, symmetriska U-tvärsnitt med styckevis konstant väggtjocklek och polärsymmetriska Z-tvärsnitt med styckevis konstant väggtjocklek. Materialet är linjärt elastiskt och balken antingen 1) gaffellagrad i båda ändar eller 2) fast inspänd i båda ändar eller 3) en konsol. Maximala spänningar och rotationer i tvärsnittet beräknas enligt S:t Venants och Vlasovs teori och lösningar approximeras med FEM. Programmet är fritt tillgängligt i pedagogiskt syfte.

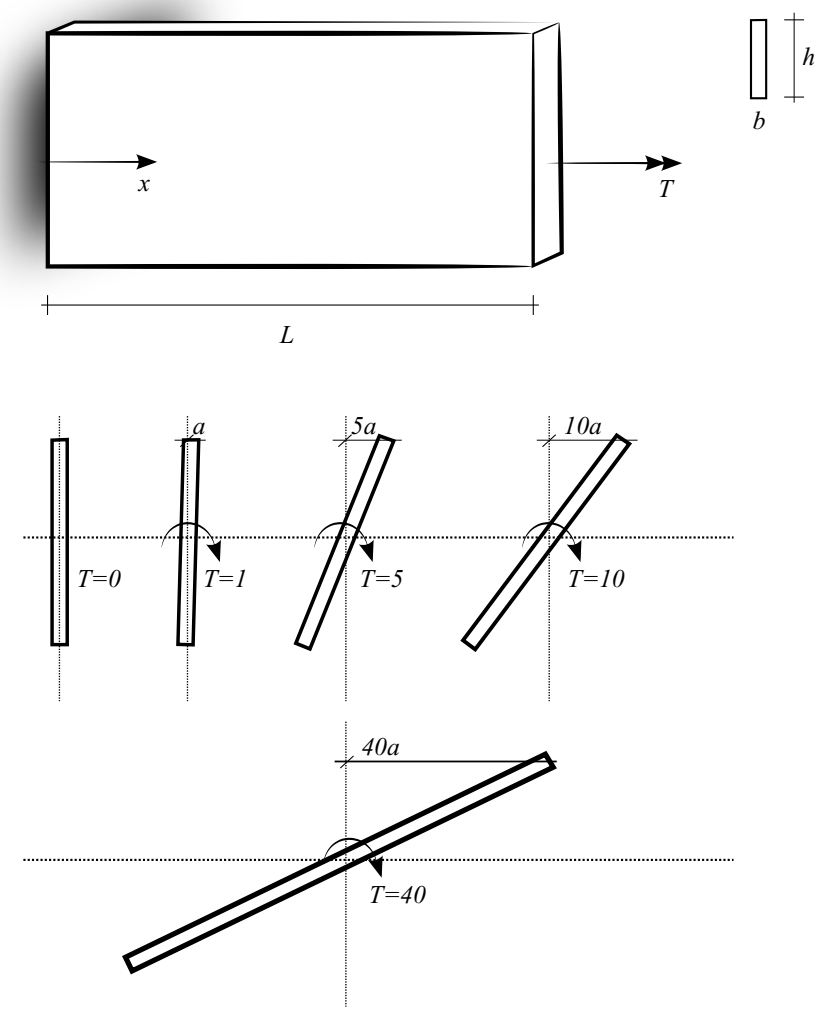
6.4 Felkällor

Beroende på hur lasten appliceras i FE-modellen för 3D-visualiseringen, uppstår ändstörningar i större eller mindre omfattning. Genom att applicera lasten i fler noder och låta lastens storlek följa skjuvspänningsfördelningen som hade uppstått i tvärsnittet enligt balkteori, hade ändstörningarna kunnat minskas.

6.5 Utvecklingsförslag

Användargränssnittet skulle kunna göras med stöd för flera tvärsnitt än de aktuella (I, U och Z). Användaren skulle i bästa fall beredas en rityta där ett valfritt tvärsnitt kunde anges. Beräkning skulle kunna göras av spänningar och rotationer enligt teorin för blandad vridning och användaren få ut information om dessa spänningar och rotationer (till exempel i diagramform) utmed balklängden. Fler sorters upplagsvillkor och lastförhållanden skulle kunna väljas.

Tänkbara utmaningar som väntar den som försöker implementera dessa förslag inbegriper bestämningen av tvärsnittets skjuvcentrum. Detta är aktuellt då tvärsnittet ej är symmetriskt. Das [5] har en användbar algoritm för att bestämma skjuvcentrums läge numeriskt. Om användaren själv kan rita tvärsnittsytans form (som inte nödvändigtvis består av tunna rektanglar med styckevis konstant väggtjocklek) är det en utmaning att bestämma vridstyvhetens tvärsnittsfaktor och välvtröghetsmomentet. När det gäller stöd för fler upplagsvillkor och lastförhållanden gäller det att bereda användaren möjlighet att påverka detta på ett så användarvänligt vis som möjligt. Om användaren till exempel ska kunna placera linjelaster och punktlaster valfritt utmed balkens längd är frågan hur detta enklast kan anordnas. Via en rullgardinsmiljö? Eller genom att dra och släppa ikoner och/eller symboler på en modellfigur?



Figur 6.4: En vridbelastad konsolbalk med ett tunnväggigt rektangulärt tvärsnitt b x h, där $b << h$. Tvärsnittets rotation vid $x = L$ beror linjärt på vridmomentets storlek enligt första ordningens teori. Förskjutningarna antas små så att $\sin\varphi \approx \varphi$.

För att få realistiska visualiseringar när deformationerna är stora krävs en olinjär beräkning. Antingen bibehåller lasten sin placering och riktning eller också följer den med balkens deformation på något sätt. Första ordningens teori förutsätter små förskjutningar så att sinus för en vinkeländring approximativt är lika med vinkeländringen själv. Detta kan leda till orealistiska resultat när förutsättningarna ej är uppfyllda, jämför Figur 6.4.

Genom att införa sparse-matriser och använda en annan lösare skulle beräkningshastigheten kunna ökas så, att visualiseringen kunde uppdateras i realtid allteftersom användaren matar in ny data eller ändrar i egenskaperna för tvärsnittet. Detta hade ökat förståelsen för de visualiserade fenomenen på ett helt annat vis.

Källor

[1] Austrell, P-E et al (2004): *CALFEM. A finite element toolbox. Version 3.4* Avdelningen för byggnadsmekanik, Lunds Universitet.

[2] CERN Bulletin (2006-07-31): "Python : the holy grail of programming". *CERN Publications.* Hämtad 2013-12-15. http://cds.cern.ch/journal/CERNBulletin/2006/31/News%20Articles/974627?ln=en

[3] Christian, Wolfgang & Belloni, Mario (2002): *Physlets: Teaching physics with interactive curricular material* Prentice Hall Inc.

[4] Danielsson, Henrik (2013): *Perpendicular to grain fracture analysis of wooden structural elements. Models and applications.* Doktorsavhandling, Rapport TVSM–1024, Avdelningen för byggnadsmekanik, Lunds Universitet

[5] Das, Santanu Kumar (2007): *Computerized numerical solutions to combined pure and warping torsion in open sections.* Masteruppsats i *Civil and environmental engineering.* Massachusetts Institute of Technology.

[6] Edholm, Andreas (2013): *Meshing and visualisation routines in the Python version of CALFEM.* Masteruppsats, TVSM–13/5187, Avdelningen för byggnadsmekanik, Lunds Universitet.

[7] Engström, Lil (2006): *Möjligheter till lärande i matematik. Lärares problemformuleringar och dynamisk programvara.* Akademisk avhandling vid Stockholms universitet.

[8] Forssell, Anna (2008): *Boken om pedagogerna* Liber AB

[9] "Organizations using Python". *Python Software Foundation.* Hämtad 2013-12-15. https://wiki.python.org/moin/OrganizationsUsingPython

[10] "General Python FAQ". Python v2.7.6 documentation. Hämtad 2013-12-15. http://docs.python.org/2/faq/general.html#why-is-it-called-python

[11] Gustafsson, Per-Johan (2012): *Kursen Balkteori. Föreläsnings-OH, övningsuppgifter och formelsammanfattning.* KFS i Lund AB.

[12] Gustavsson, Bernt (2009): *Utbildningens förändrade villkor.* Liber AB.

[13] Högskoleverkets skriftserie 1999:4 S *Datorstödd eller datorstörd matematikundervisning?*

[14] Jalote, Pankaj (2005): *A concise introduction to software engineering.* Springer Verlag London Limited.

[15] Kallin Westin, Lena & Palmquist, Lena (2001): *Från siffror till surfning. Könsperspektiv på informationsteknik* Studentlitteratur

[16] Kollbrunner & Basler (1969): *Torsion in structures*. Springer-Verlag, New York.

[17] Landin, P. J. (Mars 1966): "The next 700 programming languages". *Communications of the ACM* 9 (3): 157–166. Hämtad 2013-12-15. `http://fsl.cs.illinois.edu/images/e/ef/P157-landin.pdf`

[18] Lundahl, Christian (2014): *Bedömning för lärande*. Studentlitteratur.

[19] Martelli, Alex: *Python Cookbook* 2:a uppl., sid. 230

[20] MATLAB, Version R2012b. The MathWorks Inc.

[21] Månsson, Jonas & Nordbeck, Patrik (2013): *Flerdimensionell analys*. Studentlitteratur.

[22] Nilsson, Bertil (2013): *Finita elementmetoden. En kort introduktion till teorin.* Högskolan i Halmstad. Hämtad från: verb+http://dixon.hh.se/bertil/Kurser/Common/FEMgk/Notes/kompendiumA4.pdf+ 2014-10-14

[23] NumPy, Version 1.9.0, `http://www.numpy.org/`

[24] Wikipedia uppslagsord NumPy. Hämtad från: `http://en.wikipedia.org/wiki/NumPy` 2014-09-01

[25] Ottosen, N & Petersson, H (1992): *Introduction to the Finite Element Method* Prentice Hall, Storbritannien

[26] Ottosson, Andreas (2010), *Implementation of CALFEM for Python*. Wallin & Dalholm Digital AB, Lund.

[27] PyQt4, Version 4.10.4, Riverbank Computing Limited. `http://www.riverbankcomputing.co.uk/`

[28] Riverbank Computing Limited: *PyQt Whitepaper* Hämtad från `www.riverbankcomputing.co.uk` 2014-09-01

[29] Python, Version 3.4.1, `www.python.org`

[30] van Rossum, G (2009-01-20): "A brief timeline of Python". *The history of Python*. Hämtad 2013-12-15 `http://python-history.blogspot.se/2009/01/brief-timeline-of-python.html`

[31] Salminen-Karlsson, Minna (1999): *Bringing women into computer engineering. Curriculum reform processes at two institutes of technology*. Akademisk avhandling vid Linköpings universitet.

[32] Shafer, Daniel G. (17 January 2003): "Python Streamlines Space Shuttle Mission Design". *Python Software Foundation*. Hämtad 2013-12-15. `http://www.python.org/about/success/usa/`

[33] Spyder, Version 2.3.0rc. Pierre Raybaut. `https://code.google.com/p/spyderlib/`

[34] Säljö, R. (2000): *Lärande i praktiken. Ett sociokulturellt perspektiv.* Bokförlaget Prisma

[35] Salsburg, David (2001): *The lady tasting tea. How statistics revolutionized science in the twentieth century.* W. H. Freeman / Owl Book.

[36] Skolverket: *Läroplan, examensmål och gymnasiegemensamma ämnen för gymnasieskola 2011* http://www.skolverket.se/publikationer?id=2705

[37] Skolverket: *Ämnesplan fysik*
http://www.skolverket.se/laroplaner-amnen-och-kurser/gymnasieutbildning/gymnasieskola/fys?tos=gy&subjectCode=fys&lang=sv

[38] Skolverket: *Ämnesplan matematik*
http://www.skolverket.se/laroplaner-amnen-och-kurser/gymnasieutbildning/gymnasieskola/mat?tos=gy&subjectCode=mat&lang=sv

[39] TIOBE Software Index (2013): "TIOBE Programming Community Index Python". Hämtad 2013-12-15. http://www.tiobe.com/index.php/paperinfo/tpci/Python.html

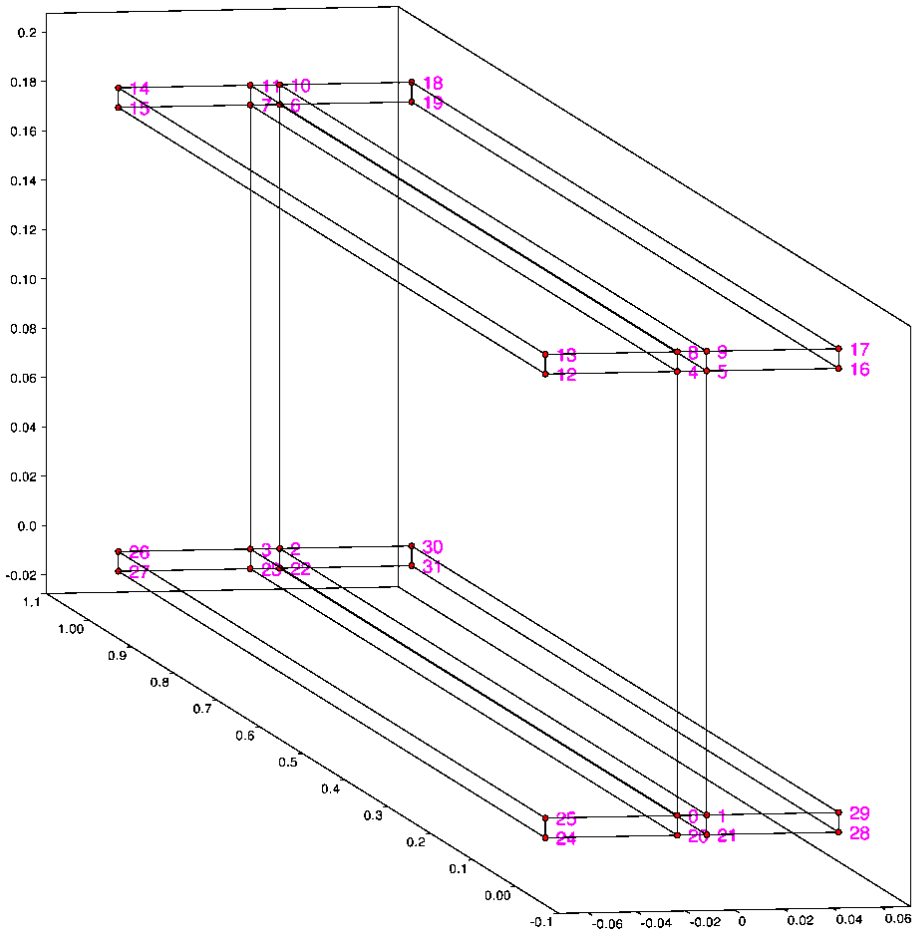
[40] The National Archives. "Annotated photographs of the COLOSSUS electronic digital computer". Foreign Office. Date: 1943. FO 850/234. Storbritannien.

[41] VanWyk, Steve (2008): *Computer solutions in physics. With applications in astrophysics, biophysics, differential equations, and engineering.* World Scientific Publishing Co. Pte. Ltd.

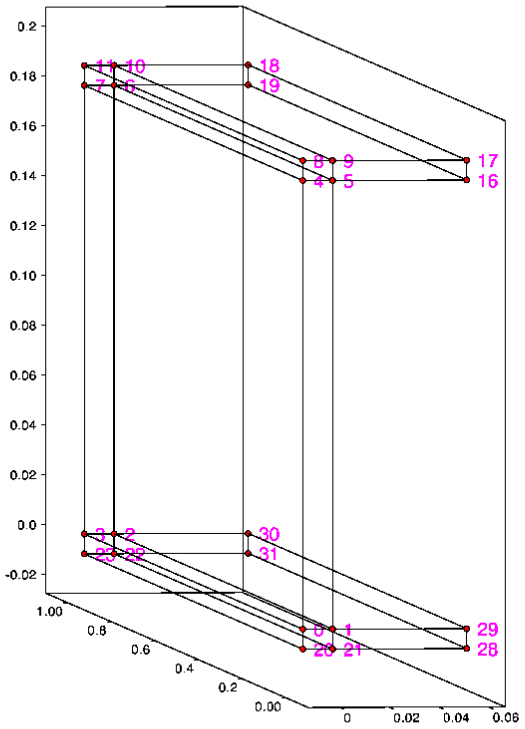
[42] Visvis, Version 1.9, https://code.google.com/p/visvis/

Bilaga 1

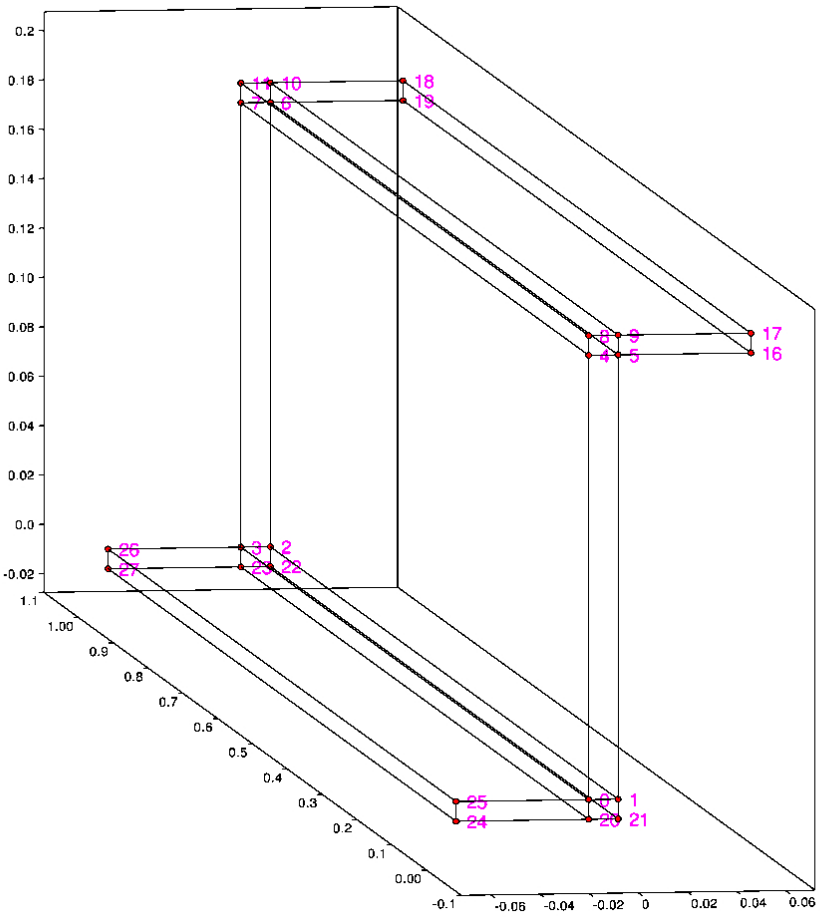
Nodnumrering i det finita elementnätet för I-, U- och Z-profilerna



Figur 6.5: Numrering av noder i 3D-finita elementnätet, I-tvärsnitt.



Figur 6.6: Numrering av noder i 3D-finita elementnätet, C-tvärsnitt.



Figur 6.7: Numrering av noder i 3D-finita elementnätet, Z-tvärsnitt.

Bilaga 2

Pythonkod

```
# -*- coding: utf-8 -*-
"""
pcv.drawDisplacements has been overridden and is located in
extrasIncludingMethodsPortedFromCALFEM together with necessary methods
"""
import os, sys, time

sys.path.append(os.path.join(os.getcwd(),"Torsion Visualiser Data"))

from extrasIncludingMethodsPortedFromCALFEM import *

try:
    from PyQt4 import QtGui, QtCore, QtSvg
    backend = 'pyqt4'
except ImportError:
    from PySide import QtGui, QtCore, QtSvg
    backend = 'pyside'

import visvis as vv

from pycalfem import *
from pycalfem_GeoData import *
from pycalfem_mesh import *
from pycalfem_utils import *
import pycalfem_vis as pcv

from numpy import *


# Create a visvis app instance, which wraps a qt4 application object.
# This needs to be done *before* instantiating the main window.
vv_app = vv.use(backend)

class Window(QtGui.QWidget):
    ''' The main window inherits from QWidget,
                                a convenient widget for an empty window. '''

    def __init__(self, parent=None):
        super(Window, self).__init__(parent)
        # Initialize the object as a QWidget and
        # set its title and minimum width.
        QtGui.QWidget.__init__(self)
        self.setWindowTitle('Torsion visualizer and calculator')
        self.setMinimumWidth(800)
        self.move(30,30)

        # Create the QVBoxLayout that lays out the whole form
        self.layout = QtGui.QVBoxLayout()

        # Create a QHBoxLayout
        self.horisontal_layout = QtGui.QHBoxLayout()

        # Create a WVBoxLayout
        self.vertical_layout = QtGui.QVBoxLayout()

        # Create the form layout that manages the labeled controls
        self.form_layout = QtGui.QFormLayout()

        # Create a place_holder for an empty row in the form layout
        self.place_holder=QtGui.QLabel('',self)

        # Insert a row with the following text
        self.form_layout.addRow('&Cross sectional properties',self.place_holder)

        # Create the items for the combo box
        self.cross_sections = ['I',
                               'C',
                               'Z']

        # Create and fill the combo box to choose the cross section
        self.cross_section = QtGui.QComboBox(self)
        self.cross_section.addItems(self.cross_sections)

        # Add it to the form layout with a label
        self.form_layout.addRow('&Cross section type', self.cross_section)

        # Connect the activated signal on the combo box to our handler.
```

```
# This is an overloaded signal, meaning there are variants of it, for
# example the activated(int) variant emits the index of the chosen
# option, rather than it's text
self.connect(self.cross_section, QtCore.SIGNAL('activated(QString)'),
                self.cross_section_chosen)

# Create a palette which will be used to color any errenous lineEdit red
self.palette = QtGui.QPalette()

# Create the entry control to specify the
# beam length and set its placeholder text
self.L = QtGui.QLineEdit(self)
self.L.setText("1e3")
# Connect the line edit with a handler in case its value changes
self.L.editingFinished.connect(lambda: self.handleEditingFinished(self.L))

# Add it to the form layout with a label
self.form_layout.addRow('&Beam length [mm]:', self.L)

# Create the entry control to specify the
# web thickness and set its placeholder text
self.web_thickness = QtGui.QLineEdit(self)
self.web_thickness.setText("6.5")
# Connect the line edit with a handler in case its value changes
self.web_thickness.editingFinished.connect(lambda:
            self.handleEditingFinished(self.web_thickness))

# Add it to the form layout with a label
self.form_layout.addRow('&Web thickness [mm]:', self.web_thickness)

# Create the entry control to specify the
# web height and set its placeholder text
self.web_height = QtGui.QLineEdit(self)
self.web_height.setText("190")
# Connect the line edit with a handler in case its value changes
self.web_height.editingFinished.connect(lambda:
                self.handleEditingFinished(self.web_height))

# Add it to the form layout with a label
self.form_layout.addRow('&Web height [mm]:', self.web_height)

# Create the entry control to specify the
# top flange thickness and set its placeholder text
self.flange_top_thickness = QtGui.QLineEdit(self)
self.flange_top_thickness.setText("10")
# Connect the line edit with a handler in case its value changes
self.flange_top_thickness.editingFinished.connect(lambda:
                self.handleEditingFinished(self.flange_top_thickness))

# Add it to the form layout with a label
self.form_layout.addRow('&Top flange thickness [mm]:',
                self.flange_top_thickness)

# Create the entry control to specify the
# top flange width and set its placeholder text
self.flange_top_width = QtGui.QLineEdit(self)
self.flange_top_width.setText("200")
# Connect the line edit with a handler in case its value changes
self.flange_top_width.editingFinished.connect(lambda:
                self.handleEditingFinished(self.flange_top_width))

# Add it to the form layout with a label
self.form_layout.addRow('&Top flange width [mm]:', self.flange_top_width)

# Create the entry control to specify the
# bottom flange thickness and set its placeholder text
self.flange_bottom_thickness = QtGui.QLineEdit(self)
self.flange_bottom_thickness.setText("10")
# Connect the line edit with a handler in case its value changes
self.flange_bottom_thickness.editingFinished.connect(lambda:
            self.handleEditingFinished(self.flange_bottom_thickness))

# Add it to the form layout with a label
self.form_layout.addRow('&Bottom flange thickness [mm]:',
                self.flange_bottom_thickness)

# Create the entry control to specify the
# bottom flange width and set its placeholder text
self.flange_bottom_width = QtGui.QLineEdit(self)
self.flange_bottom_width.setText("200")
# Connect the line edit with a handler in case its value changes
self.flange_bottom_width.editingFinished.connect(lambda:
                self.handleEditingFinished(self.flange_bottom_width))

# Add it to the form layout with a label
self.form_layout.addRow('&Bottom flange width [mm]:',
        self.flange_bottom_width)

# Add an empty row
self.form_layout.addRow('',self.place_holder)

# Add a line with the following text
self.form_layout.addRow('&Material properties',self.place_holder)

# Create the entry control to specify the
# modulus of torsion and set its placeholder text
self.G = QtGui.QLineEdit(self)
self.G.setText("81")
# Connect the line edit with a handler in case its value changes
self.G.editingFinished.connect(lambda: self.handleEditingFinished(self.G))
```

```
# Add it to the form layout with a label
self.form_layout.addRow("&Modulus of torsion [GPa]:", self.G)

# Create the entry control to specify
# Poisson's ratio and set its placeholder text
self.v = QtGui.QLineEdit(self)
self.v.setText("0.3")
# Connect the line edit with a handler in case its value changes
self.v.editingFinished.connect(lambda: self.handleEditingFinished(self.v))

# Add it to the form layout with a label
self.form_layout.addRow("&Poisson's ratio:", self.v)

# Add an empty line
self.form_layout.addRow('',self.place_holder)
# Add a line with the following text
self.form_layout.addRow('&Boundary conditions',self.place_holder)

# Create the entry control to specify the
# applied torsion and set its placeholder text
self.T = QtGui.QLineEdit(self)
self.T.setText("1")

# Connect the line edit with a handler in case its value changes
self.T.editingFinished.connect(lambda: self.handleEditingFinished(self.T))

# Add it to the form layout with a label
self.form_layout.addRow("&Applied torsion [KNm]:", self.T)

# Create the items for a combo box
self.load_types= ['Point load',
                  'Distributed load']

# Create and fill the combo box to choose the load type
self.load_type_cb = QtGui.QComboBox(self)
self.load_type_cb.addItems(self.load_types)

# Add it to the form layout with a label
self.form_layout.addRow('&Load type', self.load_type_cb)

# Connect the activated signal on the combo box to our handler.
# This is an overloaded signal, meaning there are variants of it, for
# example the activated(int) variant emits the index of the chosen
# option, rather than it's text
self.connect(self.load_type_cb, QtCore.SIGNAL('activated(QString)'),
                self.load_type_chosen)

# Add an empty line
self.form_layout.addRow('',self.place_holder)

# Create a label which holds the loading type illustration and set the image
self.bc_type = QtGui.QLabel(self)
self.bc_type.index=0
self.bc_type.PixMap = QtGui.QPixmap(os.path.join(os.getcwd(),
                "Torsion Visualiser Data", "boundary_conditions_console.png"))
self.bc_type.setScaledContents(True)
self.bc_type.setFixedWidth(200)
self.bc_type.setFixedHeight(40)
self.bc_type.setPixmap(self.bc_type.PixMap)

# Create the toggle button with its caption
self.toggle_button = QtGui.QPushButton('&Toggle', self)

# Connect the button's clicked signal to toggle handler
self.toggle_button.clicked.connect(lambda:
                self.toggle_bc_type(self.bc_type.index))

# Add it to the form layout
self.form_layout.addRow(self.bc_type, self.toggle_button)

# Add lines of white space
self.form_layout.addRow('',self.place_holder)
self.form_layout.addRow('',self.place_holder)
self.form_layout.addRow('',self.place_holder)
self.form_layout.addRow('',self.place_holder)
self.form_layout.addRow('',self.place_holder)
self.form_layout.addRow('',self.place_holder)
self.form_layout.addRow('',self.place_holder)
self.form_layout.addRow('',self.place_holder)
self.form_layout.addRow('',self.place_holder)
self.form_layout.addRow('',self.place_holder)
self.form_layout.addRow('',self.place_holder)

# Create and add the label with St Venant tension
self.St_Venant_stress = QtGui.QLabel('', self)
self.form_layout.addRow("St Venant stress:", self.St_Venant_stress)

# Create and add the label with St Venant twist
self.St_Venant_twist = QtGui.QLabel('', self)
self.form_layout.addRow("St Venant twist:", self.St_Venant_twist)

# Create and add the label with Vlasov shear stress
self.Vlasov_tau = QtGui.QLabel('', self)
self.form_layout.addRow("Vlasov shear stress:", self.Vlasov_tau)

# Create and add the label with Vlasov normal stress
self.Vlasov_sigma= QtGui.QLabel('', self)
self.form_layout.addRow("Vlasov normal stress:", self.Vlasov_sigma)
```

```
        # Create and add the label with Vlasov twist
        self.Vlasov_twist= QtGui.QLabel('', self)
        self.form_layout.addRow("Vlasov twist:", self.Vlasov_twist)

        # Add an empty line
        self.form_layout.addRow('',self.place_holder)

        # Add the form layout to the main VBox layout
        self.horisontal_layout.addLayout(self.form_layout,1)

        # Add stretch to the horisontal layout
        self.horisontal_layout.addStretch(1)

        # Create a figure for the Visvis app and add it to the layout
        Figure = vv_app.GetFigureClass()
        self.fig = Figure(self)
        self.vertical_layout.addWidget(self.fig._widget)

        # Create a horisontal layout under the Visvis app
        self.horisontal_layout_under_vis=QtGui.QHBoxLayout()

        # Add text, a line edit and a check box to the layout
        self.scroll_info_text=QtGui.QLabel(
            "Use mouse to rotate view. \nMove: + Shift. \nPan: + Ctrl. \nScroll wheel zooms")
        self.horisontal_layout_under_vis.addWidget(self.scroll_info_text)
        self.horisontal_layout_under_vis.addStretch(1)

        self.vertical_layout_under_vis1=QtGui.QVBoxLayout()
        self.vertical_layout_under_vis2=QtGui.QVBoxLayout()
        self.vertical_layout_under_vis3=QtGui.QVBoxLayout()

        self.cb_axis = QtGui.QCheckBox('Show axis', self)
        self.cb_axis.toggle()
        self.cb = QtGui.QCheckBox('Display non-deformed mesh', self)
        self.vertical_layout_under_vis1.addWidget(self.cb_axis)
        self.vertical_layout_under_vis1.addWidget(self.cb)
        self.horisontal_layout_under_vis.addLayout(self.vertical_layout_under_vis1)

        self.colorButton=QtGui.QPushButton('&Toggle background colors')
        self.vertical_layout_under_vis3.addWidget(self.colorButton)
        self.colorButton.clicked.connect(lambda: self.toggleColors(
                        self.beam.fEM.colorIndex))

        self.screenshotButton=QtGui.QPushButton('&Save screenshot.jpg')
        self.vertical_layout_under_vis3.addWidget(self.screenshotButton)
        self.screenshotButton.clicked.connect(self.screenshotAction)

        self.horisontal_layout_under_vis.addLayout(self.vertical_layout_under_vis3)

        self.horisontal_layout_under_vis.addStretch(1)

        self.magn_fact_text=QtGui.QLabel("Magnification factor: ")
        self.magn_fact_box=QtGui.QLineEdit(self)
        self.magn_fact_box.setFixedWidth(35)
        self.magn_fact_box.setText('1')
        self.horisontal_layout_under_vis2=QtGui.QHBoxLayout()
        self.horisontal_layout_under_vis2.addWidget(self.magn_fact_text)
        self.horisontal_layout_under_vis2.addWidget(self.magn_fact_box)
        self.vertical_layout_under_vis2.addLayout(self.horisontal_layout_under_vis2)


        self.new_window_button = QtGui.QPushButton(
                        '&Open visualisation in new window', self)
        self.vertical_layout_under_vis2.addWidget(self.new_window_button)
        self.new_window_button.clicked.connect(self.openNewWindow)

        self.horisontal_layout_under_vis.addLayout(self.vertical_layout_under_vis2)

        # Connect the line edit to the event handler
        self.magn_fact_box.editingFinished.connect(lambda:
                        self.changeMagnFactor(self.magn_fact_box))

        # Connect the check box to the event handler
        self.cb.stateChanged.connect(self.changeMeshDisplay)

        # Connect the check box to the event handler
        self.cb_axis.stateChanged.connect(self.changeAxis)

        # Add the layout to the parent layout
        self.vertical_layout.addLayout(self.horisontal_layout_under_vis)

        # Create an illustration of the cross section
        self.cs_illustration = QtGui.QLabel(self)
        self.cs_illustration.setScaledContents(True)
        self.cs_illustration.setFixedWidth(600)
        self.cs_illustration.setFixedHeight(300)
        self.cs_illustration.PixMap = QtGui.QPixmap(os.path.join(os.getcwd(),
            "Torsion Visualiser Data","cross_section_details_I.png"))
        self.cs_illustration.setPixmap(self.cs_illustration.PixMap)

        # Add the illustration to the layout
        self.vertical_layout.addWidget(self.cs_illustration)

#         self.overlay2 = Overlay2(self.cs_illustration)
#         self.overlay2.move(200,200)

        # Add stretch to separate the form layout from the button
        self.layout.addStretch(1)

        # Add the vertical layout to the horisontal
```

```
        self.horisontal_layout.addLayout(self.vertical_layout,2)

        # Add the horisontal layout to the parent
        self.layout.addLayout(self.horisontal_layout)

        # Add stretch to separate the form layout from the button
        self.layout.addStretch(1)

        # Create a horizontal box layout to hold the button
        self.button_box = QtGui.QHBoxLayout()

        # Add stretch to push the button to the far right
        self.button_box.addStretch(1)

        # Create a progress bar
        self.progressBar=QtGui.QProgressBar()

        # Add the progress bar to the layout
        self.button_box.addWidget(self.progressBar)

        # Hide the progress bar
        self.progressBar.hide()

        # Create the build button with its caption
        self.calculate_button = QtGui.QPushButton('&Calculate and visualize', self)

        # Add it to the button box
        self.button_box.addWidget(self.calculate_button)

        # Add the button box to the bottom of the main VBox layout
        self.layout.addLayout(self.button_box)

        # Set the VBox layout as the window's main layout
        self.setLayout(self.layout)

#         Connect the button's clicked signal to perform calculation and visualization
        self.calculate_button.clicked.connect(self.onStartThreadButtonClicked)

        # Create an instance of the Beam class
        self.beam=Beam(self.cross_section.currentText(),
            self.L.text(),
            self.web_thickness.text(),
            self.web_height.text(),
            self.flange_top_thickness.text(),
            self.flange_top_width.text(),
            self.flange_bottom_thickness.text(),
            self.flange_bottom_width.text(),
            self.G.text(),
            self.v.text(),
            self.T.text(),
            self.load_type_cb.currentText(),
            self.bc_type.index)

        # Add properties to the beam instance
        self.beam.st_Venant=St_Venant()
        self.beam.vlasov=Vlasov()
        self.beam.fEM=FEM()

        # Draw an initial 3D visualisation
        # Calculate Vlasov
        self.beam.Vlasov_calculation()
        # Calculate 3D FEM model
        self.beam.FEM()
        # Print data to GUI
        self.setData()
        # Draw 3D visualisation
        self.setVisualisation()

    def openNewWindow(self):
        self.holdCurrentView()
        vv.figure()
        drawDisplacementsWithFillOption(self.beam.fEM.a, self.beam.fEM.coords,
                        self.beam.fEM.edof, self.beam.fEM.dofsPerNode,
                    self.beam.fEM.elType,
                        doDrawUndisplacedMesh=self.beam.fEM.undeformedMesh,
                    title='', filled=True, magnfac=self.beam.fEM.mf)
        vv.view(self.beam.fEM.viewProperties)
        vv.gca().axis.showBox=False
        if self.beam.fEM.show_axis==True:
            vv.gca().axis.visible=1
        else:
            vv.gca().axis.visible=0
        vv.xlabel('Width [m]')
        vv.ylabel('Length [m]')
        vv.zlabel('Height [m]')
        vv.gca().bgcolors=self.beam.fEM.bgcolors

    def changeAxis(self, state):
        if state == QtCore.Qt.Checked:
            self.beam.fEM.show_axis=True
        else:
            self.beam.fEM.show_axis=False

        self.holdCurrentView()
        self.setVisualisation()

    def holdCurrentView(self):
        self.beam.fEM.viewProperties=vv.view()

    def changeMeshDisplay(self, state):
```

```
        if state == QtCore.Qt.Checked:
            self.beam.fEM.undeformedMesh=True
        else:
            self.beam.fEM.undeformedMesh=False

        self.holdCurrentView()
        self.setVisualisation()

    def changeMagnFactor(self, flag):

        if flag.isModified():
            self.palette.setColor(flag.backgroundRole(), QtGui.QColor('white'))
            flag.setPalette(self.palette)
            try:
                self.beam.fEM.mf=float(flag.text())
            except ValueError:
                QtGui.QMessageBox.critical(self,
                    "Message", "Magnification factor: please enter a valid number!")
                self.palette.setColor(flag.backgroundRole(), QtGui.QColor('red'))
                flag.setPalette(self.palette)
                flag.setText("1")
                self.beam.fEM.mf=1
        #Read and save the current view properties
        self.holdCurrentView()
        self.setVisualisation()
        flag.setModified(False)

    def onUpdate(self, txt):
        """Update UI for calculation status here."""
        n=10
        self.progressBar.setValue(self.progressBar.value()+1*n)
        if self.progressBar.value()==2*n:
            self.beam.FEM()
        elif self.progressBar.value()==3*n and self.beam.bc_type==1:
            self.beam.St_Venant_calculation()
        elif self.progressBar.value()==4*n and self.beam.bc_type!=1:
            self.beam.Vlasov_calculation()
        elif self.progressBar.value()==5*n:
            self.setData()
        elif self.progressBar.value()==6*n:
            self.holdCurrentView()
        elif self.progressBar.value()==7*n:
            self.setVisualisation()

    def onCompleted(self):
        """Handle completion of calculation thread here."""
        self.calculate_button.setEnabled(True)
        self.progressBar.hide()

    def onStartThreadButtonClicked(self):
        """Start calculation thread. Connect our signals to methods in the class."""
        self.workThread = WorkThread()
        self.progressBar.show()
        self.progressBar.setValue(0)
        self.connect( self.workThread, QtCore.SIGNAL("update(QString)"), self.onUpdate )
        self.connect( self.workThread, QtCore.SIGNAL("completed()"), self.onCompleted )
        self.calculate_button.setEnabled(False)
        self.workThread.start()

    def setData(self):
        if self.beam.bc_type==1:
            self.St_Venant_stress.setText(str(round(self.beam.st_Venant.tau_max,3))+
                                self.beam.st_Venant.tau_unit)
            self.St_Venant_twist.setText(str(round(self.beam.st_Venant.rot_max,3))+
                                self.beam.st_Venant.rot_unit)
            self.Vlasov_sigma.setText("-")
            self.Vlasov_tau.setText("-")
            self.Vlasov_twist.setText("-")

        elif self.beam.bc_type==0 or self.beam.bc_type==2:
            self.St_Venant_stress.setText("-")
            self.St_Venant_twist.setText("-")
            self.Vlasov_tau.setText(str(round(self.beam.vlasov.tau_max,3))+
                                self.beam.vlasov.tau_unit)
            self.Vlasov_sigma.setText(str(round(self.beam.vlasov.sigma_max,3))+
                                self.beam.vlasov.sigma_unit)
            self.Vlasov_twist.setText(str(round(self.beam.vlasov.rot_max,3))+
                                self.beam.vlasov.rot_unit)

    def screenshotAction(self):

        self.holdCurrentView()
        vv.figure(2).position = 0,0,2000,2000
        drawDisplacementsWithFillOption(self.beam.fEM.a, self.beam.fEM.coords,
                        self.beam.fEM.edof, self.beam.fEM.dofsPerNode,
                self.beam.fEM.elType,
                        doDrawUndisplacedMesh=self.beam.fEM.undeformedMesh,
                        title='', filled=True, magnfac=self.beam.fEM.mf)
        vv.view(self.beam.fEM.viewProperties)
        vv.gca().axis.showBox=False
        if self.beam.fEM.show_axis==True:
            vv.gca().axis.visible=1
        else:
            vv.gca().axis.visible=0
        vv.gca().bgcolors=self.beam.fEM.bgcolors
        vv.screenshot('screenshot.jpg', vv.gca(), sf=3, bg='w')
        vv.close(2)
```

```
    def toggleColors(self, index):
        if index==0:
            self.beam.fEM.colorIndex=1
            self.beam.fEM.bgcolors=[[.9,.9,.9],[.3,.5,.7]]
        elif index==1:
            self.beam.fEM.colorIndex=2
            self.beam.fEM.bgcolors=[[.7,.7,.7],[.7,.7,.7]]
        elif index==2:
            self.beam.fEM.colorIndex=0
            self.beam.fEM.bgcolors=['w', 'w']
        self.holdCurrentView()
        self.setVisualisation()

    def setVisualisation(self):
        vv.figure(self.fig)
        vv.clf()
        drawDisplacementsWithFillOption(self.beam.fEM.a, self.beam.fEM.coords,
                    self.beam.fEM.edof, self.beam.fEM.dofsPerNode,
                        self.beam.fEM.elType,
                    doDrawUndisplacedMesh=self.beam.fEM.undeformedMesh,
                        title='', filled=True, magnfac=self.beam.fEM.mf)
        vv.view(self.beam.fEM.viewProperties)
        vv.gca().axis.showBox = False
        if self.beam.fEM.show_axis==True:
            vv.gca().axis.visible=1
        else:
            vv.gca().axis.visible=0
        vv.xlabel('Width [m]')
        vv.ylabel('Length [m]')
        vv.zlabel('Height [m]')
        vv.gca().bgcolors=self.beam.fEM.bgcolors

    def cross_section_chosen(self, text):
        """
        Handler called when a distro is chosen from the combo box
        """
        if text=='I':
                self.cs_illustration.PixMap = QtGui.QPixmap(os.path.join(os.getcwd(),
                "Torsion Visualiser Data", "cross_section_details_I.png"))
                self.cs_illustration.setPixmap(self.cs_illustration.PixMap)

                self.flange_bottom_width.setEnabled(True)
                self.flange_bottom_thickness.setEnabled(True)

                self.flange_top_thickness.textChanged.disconnect()
                self.flange_top_width.textChanged.disconnect()

                self.beam.cross_section_type='I'

        elif text=='C':
                self.cs_illustration.PixMap = QtGui.QPixmap(os.path.join(os.getcwd(),
                "Torsion Visualiser Data", "cross_section_details_C.png"))
                self.cs_illustration.setPixmap(self.cs_illustration.PixMap)

                self.flange_bottom_width.setEnabled(False)
                self.flange_bottom_thickness.setEnabled(False)

                self.flange_top_thickness.textChanged.connect(
                    self.on_text_changed_bottom_flange)
                self.flange_top_width.textChanged.connect(
                    self.on_text_changed_bottom_flange)

                self.beam.cross_section_type='C'

        else:
                self.cs_illustration.PixMap = QtGui.QPixmap(os.path.join(os.getcwd(),
                "Torsion Visualiser Data", "cross_section_details_Z.png"))
                self.cs_illustration.setPixmap(self.cs_illustration.PixMap)

                self.flange_bottom_width.setEnabled(False)
                self.flange_bottom_thickness.setEnabled(False)

                self.flange_top_thickness.textChanged.connect(
                    self.on_text_changed_bottom_flange)
                self.flange_top_width.textChanged.connect(
                    self.on_text_changed_bottom_flange)

                self.beam.cross_section_type='Z'

    def load_type_chosen(self,text):
        """
        Handler called when a distro is chosen from the combo box
        """
        self.beam.load_type=text

    def on_text_changed_bottom_flange(self):
        self.flange_bottom_thickness.setText(self.flange_top_thickness.text())
        self.flange_bottom_width.setText(self.flange_top_width.text())
        self.handleEditingFinished(self.flange_bottom_thickness)
        self.handleEditingFinished(self.flange_bottom_width)

    def toggle_bc_type(self,index):
        if index==0:
            self.bc_type.index=1
            self.beam.bc_type=1
            self.bc_type.PixMap=QtGui.QPixmap(os.path.join(os.getcwd(),
                "Torsion Visualiser Data", "boundary_conditions_double_fork.png"))
            self.bc_type.setPixmap(self.bc_type.PixMap)
        if index==1:
```

```
            self.bc_type.index=2
            self.beam.bc_type=2
            self.bc_type.PixMap=QtGui.QPixmap(os.path.join(os.getcwd(),
                "Torsion Visualiser Data", "boundary_conditions_double_wall.png"))
            self.bc_type.setPixmap(self.bc_type.PixMap)
        if index==2:
            self.bc_type.index=0
            self.beam.bc_type=0
            self.bc_type.PixMap=QtGui.QPixmap(os.path.join(os.getcwd(),
                "Torsion Visualiser Data", "boundary_conditions_console.png"))
            self.bc_type.setPixmap(self.bc_type.PixMap)

    def handleEditingFinished(self,flag):
        if flag.isModified():
            self.palette.setColor(flag.backgroundRole(), QtGui.QColor('white'))
            flag.setPalette(self.palette)

            # Convert data to SI-units and check floats
            try:
                if flag==self.L:
                    self.beam.L=float(flag.text())/1e3
                if flag==self.web_thickness:
                    self.beam.web_thickness=float(flag.text())/1e3
                if flag==self.web_height:
                    self.beam.web_height=float(flag.text())/1e3
                if flag==self.flange_top_thickness:
                    self.beam.flange_top_thickness=float(flag.text())/1e3
                if flag==self.flange_top_width:
                    self.beam.flange_top_width=float(flag.text())/1e3
                if flag==self.flange_bottom_thickness:
                    self.beam.flange_bottom_thickness=float(flag.text())/1e3
                if flag==self.flange_bottom_width:
                    self.beam.flange_bottom_width=float(flag.text())/1e3
                if flag==self.G:
                    self.beam.G=float(flag.text())*1e9
                if flag==self.v:
                    self.beam.v=float(flag.text())
                if flag==self.T:
                    self.beam.T=float(flag.text())*1e3

            except ValueError:
                if flag==self.L:
                    QtGui.QMessageBox.critical(self, "Message",
                        "Beam length: please enter a valid number!")
                    self.palette.setColor(flag.backgroundRole(), QtGui.QColor('red'))
                    flag.setPalette(self.palette)
                    flag.setText("1000")
                if flag==self.web_thickness:
                    QtGui.QMessageBox.critical(self, "Message",
                        "Web thickness: please enter a valid number!")
                    self.palette.setColor(flag.backgroundRole(), QtGui.QColor('red'))
                    flag.setPalette(self.palette)
                    flag.setText("12")
                if flag==self.web_height:
                    QtGui.QMessageBox.critical(self, "Message",
                        "Web height: please enter a valid number!")
                    self.palette.setColor(flag.backgroundRole(), QtGui.QColor('red'))
                    flag.setPalette(self.palette)
                    flag.setText("180")
                if flag==self.flange_top_thickness:
                    QtGui.QMessageBox.critical(self, "Message",
                        "Flange thickness: please enter a valid number!")
                    self.palette.setColor(flag.backgroundRole(), QtGui.QColor('red'))
                    flag.setPalette(self.palette)
                    flag.setText("8")
                if flag==self.flange_top_width:
                    QtGui.QMessageBox.critical(self, "Message",
                        "Flange width: please enter a valid number!")
                    self.palette.setColor(flag.backgroundRole(), QtGui.QColor('red'))
                    flag.setPalette(self.palette)
                    flag.setText("120")
                if flag==self.flange_bottom_thickness:
                    QtGui.QMessageBox.critical(self, "Message",
                        "Flange thickness: please enter a valid number!")
                    self.palette.setColor(flag.backgroundRole(), QtGui.QColor('red'))
                    flag.setPalette(self.palette)
                    flag.setText("8")
                if flag==self.flange_bottom_width:
                    QtGui.QMessageBox.critical(self, "Message",
                        "Flange width: please enter a valid number!")
                    self.palette.setColor(flag.backgroundRole(), QtGui.QColor('red'))
                    flag.setPalette(self.palette)
                    flag.setText("120")
                if flag==self.G:
                    QtGui.QMessageBox.critical(self, "Message",
                        "Modulus of torsion: please enter a valid number!")
                    self.palette.setColor(flag.backgroundRole(), QtGui.QColor('red'))
                    flag.setPalette(self.palette)
                    flag.setText("81")
                if flag==self.v:
                    QtGui.QMessageBox.critical(self, "Message",
                        "Poisson's ratio: please enter a valid number!")
                    self.palette.setColor(flag.backgroundRole(), QtGui.QColor('red'))
                    flag.setPalette(self.palette)
                    flag.setText("0.3")
                if flag==self.T:
                    QtGui.QMessageBox.critical(self, "Message",
                        "Torsional moment: please enter a valid number!")
                    self.palette.setColor(flag.backgroundRole(), QtGui.QColor('red'))
```

```
                    flag.setPalette(self.palette)
                    flag.setText("15")

            flag.setModified(False)

    def run(self):
        # Show the form
                self.show()
        # Run the qt application
                qt_app.exec_()

class Beam():
        def __init__(self,cross_section_type,L,web_thickness,web_height,flange_top_thickness,
                flange_top_width,flange_bottom_thickness,
        flange_bottom_width,G,v,T,load_type,bc_type):

                self.cross_section_type=cross_section_type
                self.L=float(L)/1e3
                self.web_thickness=float(web_thickness)/1e3
                self.web_height=float(web_height)/1e3
                self.flange_top_thickness=float(flange_top_thickness)/1e3
                self.flange_top_width=float(flange_top_width)/1e3
                self.flange_bottom_thickness=float(flange_bottom_thickness)/1e3
                self.flange_bottom_width=float(flange_bottom_width)/1e3
                self.G=float(G)*1e9
                self.T=float(T)*1e3
                self.load_type=load_type
                self.bc_type=bc_type
                self.v=v

        def get_value_unit(self,value1,string):
                # Method that converts numbers and strings into powers of 10's
                if value1<=1e-6:
                        unit1=' x 1e-6 '+string
                        value1=value1*1e6
                elif value1>1e-6 and value1<=1e-3:
                        unit1=' x 1e-3 '+string
                        value1=value1*1e3
                elif value1<1e3 and value1>=1e-3:
                        unit1=' '+string
                elif value1<1e6 and value1>=1e3:
                        unit1=' k'+string
                        value1=value1/1e3
                elif value1<1e9 and value1>=1e6:
                        unit1=' M'+string
                        value1=value1/1e6
                elif value1>=1e9:
                        unit1=' G'+string
                        value1=value1/1e9

                return value1, unit1

        def St_Venant_calculation(self):
                g = GeoData()

                L=float(self.L)
                tw=float(self.web_thickness)
                hw=float(self.web_height)
                tft=float(self.flange_top_thickness)
                wft=float(self.flange_top_width)
                tfb=float(self.flange_bottom_thickness)
                wfb=float(self.flange_bottom_width)
                G=float(self.G)
                T=float(self.T)

                #Meshing properties
                resolution=16
                elOnCurve=resolution

       # Take symmetry into account
                if self.bc_type!=0:
                        L=L/2
                        T=T/2

                #Add Points:
                g.addPoint([0, 0], ID=0, marker=11)
                g.addPoint([L, 0], 1, marker=12)

                #Add spline:
                g.addSpline([0, 1], 0, elOnCurve = elOnCurve, marker=100)

                elType = 1
                dofsPerNode= 1 #Degrees of freedom per node.

                mesher = GmshMesher(geoData = g,
                                    gmshExecPath = None,
                                    elType = elType,
                                    dofsPerNode= dofsPerNode)

                #Mesh the geometry:
                coords, edof, dofs, bdofs, _ = mesher.create()

                L=L/resolution
                if self.cross_section_type=='I':
                        Kv=1.0/3*(tw**3*hw+tft**3*wft+tfb**3*wfb)
                        Wv=Kv/max(tw,tft,tfb)
                else:
                        Kv=1.0/3*(tw**3*hw+2*tft**3*wft)
                        Wv=Kv/max(tw,tft)
```

```
                ep=G*Kv/L

                #Assembling system matrix...
                nDofs = size(dofs)
                ex, ey = coordxtr(edof, coords, dofs)
                K = asmatrix(zeros([nDofs,nDofs]))

                # ----- Create element stiffness matrices Ke and assemble into K -
                for eltopo in edof:
                        Ke = bar1e(ep)
                        assem(eltopo,K,Ke)

                #Solving equation system...
                f = zeros([nDofs,1])

                bc = array([],'i')
                bcVal = array([],'i')

                bc, bcVal = applybc(bdofs, bc, bcVal, 11, 0.0, 0)

                if self.load_type=='Point load':
                        applyforce(bdofs, f, 12, T, 1)
                else:
                        f[1::3]=T/(resolution+1)

                a,r = solveq(K,f,bc,bcVal)

                #Computing element forces...
                ed = extractEldisp(edof,a)

                self.st_Venant.tau_max=T/Wv
                self.st_Venant.rot_max=amax(abs(ed))
                self.st_Venant.tau_max,self.st_Venant.tau_unit=self.get_value_unit(
                self.st_Venant.tau_max,'Pa')
                self.st_Venant.rot_max,self.st_Venant.rot_unit=self.get_value_unit(
            self.st_Venant.rot_max,'')

        def Vlasov_calculation(self):

                g = GeoData()

                L=float(self.L)
                tw=float(self.web_thickness)
                hw=float(self.web_height)
                tft=float(self.flange_top_thickness)
                wft=float(self.flange_top_width)
                tfb=float(self.flange_bottom_thickness)
                wfb=float(self.flange_bottom_width)
                G=float(self.G)
                T=float(self.T)

                #Meshing properties
                resolution=16
                elOnCurve=resolution

        # Take symmetry into account
                if self.bc_type!=0:
                        L=L/2
                        T=T/2

                #Add Points:
                g.addPoint([0, 0], ID=0, marker=11)
                g.addPoint([L, 0], 1, marker=12)

                #Add spline:
                g.addSpline([0, 1], 0, elOnCurve = elOnCurve, marker=100)

                elType = 1
                dofsPerNode= 3 #Degrees of freedom per node.

                mesher = GmshMesher(geoData = g,
                                    gmshExecPath = None,
                                    elType = elType,
                                    dofsPerNode= dofsPerNode)

                #Mesh the geometry:
                coords, edof, dofs, bdofs, _ = mesher.create()

#               Check which type of cross section
#               Formulas from Handboken Bygg 1983
                if self.cross_section_type=='I':
                        L=L/resolution
                        I_top=tft*wft**3/12
                        I_bottom=tfb*wfb**3/12
                        h=(hw+tft/2+tfb/2)
                        h_top=h*(I_bottom/(I_top+I_bottom))
                        h_bottom=h-h_top
                        I_omega=I_top*h_top**2+I_bottom*h_bottom**2
                        omega_top=h_top*wft/2
                        omega_bottom=h_bottom*wfb/2
                        omega_max=max(omega_top,omega_bottom)
                        S_top=wft**2*h_top*tft/8
                        S_bottom=wfb**2*h_bottom*tfb/8
                        S_max=max(S_top,S_bottom)
                        A=tw*hw+tft*wft+tfb*wfb

                elif self.cross_section_type=='C':
                        L=L/resolution
                        e=3*wft**2*tft/(6*wft*tft+hw*tw)
                        I_omega=wft**2*hw**2*tft*(2*wft-3*e)/12
```

```
                        omega_w=e*hw/2
                        omega_f=(wft-e)*hw/2
                        omega_max=max(omega_w,omega_f)
                        S_1=(wft-e)**2*hw*tft/4
                        S_2=(wft-2*e)*wft*hw*tft/4
                        S_3=e*hw**2*tw/8-S_2
                        S_max=max(S_1,S_2,S_3)
                        A=tw*hw+tft*wft+tfb*wfb

                elif self.cross_section_type=='Z':
                        L=L/resolution
                        e=wft**2*tft/(2*wft*tft+hw*tw)
                        I_omega=wft**2*hw**2*tft*(2*wft-3*e)/12
                        omega_w=e*hw/2
                        omega_f=(wft-e)*hw/2
                        omega_max=max(omega_w,omega_f)
                        S_f=(wft-e)**2*hw*tft/4
                        S_w=(wft-2*e)*wft*hw*tft/4
                        S_max=max(S_w,S_f)
                        A=tw*hw+tft*wft+tfb*wfb

                ep=[G,A,I_omega]

#               print "Assembling system matrix..."
                nDofs = size(dofs)
                ex, ey = coordxtr(edof, coords, dofs)
                K = asmatrix(zeros([nDofs,nDofs]))

                # ------ Create element stiffness matrices Ke and assemble into K -

                for elx, ely, eltopo in zip(ex,ey,edof):
                        Ke = beam2e(elx,ely,ep)
                        assem(eltopo,K,Ke)

#               print "Solving equation system..."
                f = zeros([nDofs,1])

                bc = array([],'i')
                bcVal = array([],'i')

                # Check boundary conditions
                if self.bc_type==0:
                        # Console
                        bc, bcVal = applybc(bdofs, bc, bcVal, 11, 0.0, 0)
                        # Check load type
                        if self.load_type=='Point load':
                                applyforce(bdofs, f, 12, -T, 2)
                        else:
                                f[1::3]=-T/(resolution+1)

                elif self.bc_type==2:
#                       # Fixed
                        bc, bcVal = applybc(bdofs, bc, bcVal, 11, 0.0, 0)
                        bc, bcVal = applybc(bdofs, bc, bcVal, 12, 0.0, 1)
                        # Check load type
                        if self.load_type=='Point load':
                                applyforce(bdofs, f, 12, -T, 2)
                        else:
                                f[1::3]=-T/(resolution+1)

                a,r = solveq(K,f,bc,bcVal)

#               print "Computing element forces..."
                Ed = extractEldisp(edof,a)

                es=zeros([1,3])
                ed=zeros([1,2])
                ec=zeros([1,1])
                for i in range(resolution):
                        es1,ed1,ec1=beam2s(ex[i,:],ey[i,:],ep,Ed[i,:])
                        es=vstack((es,es1))
                        ed=vstack((ed,ed1))
                        ec=vstack((ec,ec1))
                delete(es,0,0)
                delete(ed,0,0)
                delete(ec,0,0)

                ed_max=amax(abs(ed))
                tau_max=S_max/min(tft,tfb,tw)/I_omega*amax(abs(es[1::3]))
                sigma_max=omega_max/I_omega*amax(abs(es[2::3]))

                self.vlasov.tau_max=tau_max
                self.vlasov.sigma_max=sigma_max
                self.vlasov.rot_max=ed_max
                self.vlasov.tau_max,self.vlasov.tau_unit=self.get_value_unit(
                self.vlasov.tau_max,'Pa')
                self.vlasov.sigma_max,self.vlasov.sigma_unit=self.get_value_unit(
                    self.vlasov.sigma_max,'Pa')
                self.vlasov.rot_max,self.vlasov.rot_unit=self.get_value_unit(
                    self.vlasov.rot_max,'')

        def FEM(self):
                g = GeoData()

                L=float(self.L)
                tw=float(self.web_thickness)
                hw=float(self.web_height)
                tft=float(self.flange_top_thickness)
                wft=float(self.flange_top_width)
                tfb=float(self.flange_bottom_thickness)
```

```
wfb=float(self.flange_bottom_width)
G=float(self.G)
v=float(self.v)
T=float(self.T)

# Take symmetry into account
if self.bc_type!=0:
        L=L/2
        T=T/2

# Define P, the statically equivalent force to the torsion
P=T/hw

#Meshing properties
resolution=1
elOnCurve=resolution
elOnCurveLength=25
elOnCurveFlange=5
elOnCurveWeb=15

if self.cross_section_type=='I':
        #Add Points for Web:
        g.addPoint([-tw/2, 0, 0], ID=0, marker=11)
        g.addPoint([tw/2, 0, 0], 1)
        g.addPoint([tw/2, L, 0], 2, marker=101)
        g.addPoint([-tw/2, L, 0], 3)
        g.addPoint([-tw/2, 0, hw], 4)
        g.addPoint([tw/2, 0, hw], 5, marker = 44)
        g.addPoint([tw/2, L, hw], 6)
        g.addPoint([-tw/2, L, hw], 7)

        #Add Points for Top Flange:
        #Top middle
        g.addPoint([-tw/2, 0, hw+tft], 8)
        g.addPoint([tw/2, 0, hw+tft], 9)
        g.addPoint([tw/2, L, hw+tft], 10)
        g.addPoint([-tw/2, L, hw+tft], 11)

        #Top left
        g.addPoint([-wft/2, 0, hw], 12)
        g.addPoint([-wft/2, 0, hw+tft], 13)
        g.addPoint([-wft/2, L, hw+tft], 14)
        g.addPoint([-wft/2, L, hw], 15)

        #Top right
        g.addPoint([wft/2, 0, hw], 16, marker=55)
        g.addPoint([wft/2, 0, hw+tft], 17)
        g.addPoint([wft/2, L, hw+tft], 18)
        g.addPoint([wft/2, L, hw], 19)

        #Add Points for Bottom Flange:
        #Bottom middle
        g.addPoint([-tw/2, 0, -tfb], 20)
        g.addPoint([tw/2, 0, -tfb], 21)
        g.addPoint([tw/2, L, -tfb], 22)
        g.addPoint([-tw/2, L, -tfb], 23)

        #Bottom left
        g.addPoint([-wfb/2, 0, 0], 25, marker=66)
        g.addPoint([-wfb/2, 0, -tfb], 24)
        g.addPoint([-wfb/2, L, -tfb], 27)
        g.addPoint([-wfb/2, L, 0], 26)

        #Bottom right
        g.addPoint([wfb/2, 0, 0], 29)
        g.addPoint([wfb/2, 0, -tfb], 28)
        g.addPoint([wfb/2, L, -tfb], 31)
        g.addPoint([wfb/2, L, 0], 30)


        #Add splines for Web:
        g.addSpline([0, 1], 0, elOnCurve = elOnCurve)
        g.addSpline([1, 2], 1, elOnCurve = elOnCurveLength)
        g.addSpline([2, 3], 2, elOnCurve = elOnCurve)
        g.addSpline([3, 0], 3, elOnCurve = elOnCurveLength, marker=12)
        g.addSpline([0, 4], 4, elOnCurve = elOnCurveWeb)
        g.addSpline([1, 5], 5, elOnCurve = elOnCurveWeb)
        g.addSpline([2, 6], 6, elOnCurve = elOnCurveWeb)
        g.addSpline([3, 7], 7, elOnCurve = elOnCurveWeb)
        g.addSpline([4, 5], 8, elOnCurve = elOnCurve)
        g.addSpline([5, 6], 9, elOnCurve = elOnCurveLength, marker=45)
        g.addSpline([6, 7], 10, elOnCurve = elOnCurve)
        g.addSpline([7, 4], 11, elOnCurve = elOnCurveLength)

        #Add splines for Top Flange:
        #Top middle
        g.addSpline([8, 9], 20, elOnCurve = elOnCurve)
        g.addSpline([9, 10], 21, elOnCurve = elOnCurveLength)
        g.addSpline([10, 11], 22, elOnCurve = elOnCurve)
        g.addSpline([11, 8], 23, elOnCurve = elOnCurveLength)
        g.addSpline([4, 8], 24, elOnCurve = elOnCurve)
        g.addSpline([5, 9], 25, elOnCurve = elOnCurve)
        g.addSpline([6, 10], 26, elOnCurve = elOnCurve)
        g.addSpline([7, 11], 27, elOnCurve = elOnCurve)

        #Top left
        g.addSpline([12, 13], 28, elOnCurve = elOnCurve)
        g.addSpline([13, 14], 29, elOnCurve = elOnCurveLength)
        g.addSpline([14, 15], 30, elOnCurve = elOnCurve)
        g.addSpline([15, 12], 31, elOnCurve = elOnCurveLength)
```

```
g.addSpline([12, 4], 32, elOnCurve = elOnCurveFlange)
g.addSpline([13, 8], 33, elOnCurve = elOnCurveFlange)
g.addSpline([14, 11], 34, elOnCurve = elOnCurveFlange)
g.addSpline([15, 7], 35, elOnCurve = elOnCurveFlange)

#Top right
g.addSpline([16, 17], 36, elOnCurve = elOnCurve)
g.addSpline([17, 18], 37, elOnCurve = elOnCurveLength)
g.addSpline([18, 19], 38, elOnCurve = elOnCurve)
g.addSpline([19, 16], 39, elOnCurve = elOnCurveLength)
g.addSpline([16, 5], 40, elOnCurve = elOnCurveFlange)
g.addSpline([17, 9], 41, elOnCurve = elOnCurveFlange)
g.addSpline([18, 10], 42, elOnCurve = elOnCurveFlange)
g.addSpline([19, 6], 43, elOnCurve = elOnCurveFlange)

#Add splines for Bottom Flange:
#Bottom middle
g.addSpline([20, 21], 50, elOnCurve = elOnCurve)
g.addSpline([21, 22], 51, elOnCurve = elOnCurveLength)
g.addSpline([22, 23], 52, elOnCurve = elOnCurve)
g.addSpline([23, 20], 53, elOnCurve = elOnCurveLength)
g.addSpline([0, 20], 54, elOnCurve = elOnCurve)
g.addSpline([21, 1], 55, elOnCurve = elOnCurve)
g.addSpline([22, 2], 56, elOnCurve = elOnCurve)
g.addSpline([23, 3], 57, elOnCurve = elOnCurve)

#Bottom left
g.addSpline([24, 25], 58, elOnCurve = elOnCurve)
g.addSpline([25, 26], 59, elOnCurve = elOnCurveLength)
g.addSpline([26, 27], 60, elOnCurve = elOnCurve)
g.addSpline([27, 24], 61, elOnCurve = elOnCurveLength)
g.addSpline([24, 20], 62, elOnCurve = elOnCurveFlange)
g.addSpline([25, 0], 63, elOnCurve = elOnCurveFlange)
g.addSpline([26, 3], 64, elOnCurve = elOnCurveFlange)
g.addSpline([27, 23], 65, elOnCurve = elOnCurveFlange)

#Bottom right
g.addSpline([28, 29], 66, elOnCurve = elOnCurve)
g.addSpline([29, 30], 67, elOnCurve = elOnCurveLength)
g.addSpline([30, 31], 68, elOnCurve = elOnCurve)
g.addSpline([31, 28], 69, elOnCurve = elOnCurveLength)
g.addSpline([21, 28], 70, elOnCurve = elOnCurveFlange)
g.addSpline([1, 29], 71, elOnCurve = elOnCurveFlange)
g.addSpline([2, 30], 72, elOnCurve = elOnCurveFlange)
g.addSpline([22, 31], 73, elOnCurve = elOnCurveFlange)


#Add surfaces for Web:
g.addStructuredSurface([0, 1, 2, 3], 0)
g.addStructuredSurface([8, 9, 10, 11], 1)
g.addStructuredSurface([0, 4, 8, 5], 2, marker=200)
g.addStructuredSurface([1, 5, 9, 6], 3)
g.addStructuredSurface([2, 6, 10, 7], 4, marker=100)
g.addStructuredSurface([3, 4, 11, 7], 5)

#Add surfaces for Top Flange:
#Top middle
g.addStructuredSurface([20, 21, 22, 23], 6)
g.addStructuredSurface([24, 20, 25, 8], 7, marker=200)
g.addStructuredSurface([25, 21, 26, 9], 8)
g.addStructuredSurface([26, 22, 27, 10], 9, marker=100)
g.addStructuredSurface([27, 23, 24, 11], 10)

#Top left
g.addStructuredSurface([28, 29, 30, 31], 11)
g.addStructuredSurface([32, 24, 33, 28], 12, marker=200)
g.addStructuredSurface([33, 23, 34, 29], 13)
g.addStructuredSurface([34, 30, 35, 27], 14, marker=100)
g.addStructuredSurface([32, 11, 35, 31], 15)

#Top right
g.addStructuredSurface([36, 37, 38, 39], 16)
g.addStructuredSurface([40, 25, 41, 36], 17, marker=200)
g.addStructuredSurface([41, 21, 42, 37], 18)
g.addStructuredSurface([42, 38, 43, 26], 19, marker=100)
g.addStructuredSurface([40, 9, 43, 39], 20)

#Add surfaces for Bottom Flange:
#Bottom middle
g.addStructuredSurface([50, 51, 52, 53], 21)
g.addStructuredSurface([0, 54, 50, 55], 22, marker=200)
g.addStructuredSurface([55, 1, 56, 51], 23)
g.addStructuredSurface([2, 57, 52, 56], 24, marker=100)
g.addStructuredSurface([53, 54, 3, 57], 25)

#Bottom left
g.addStructuredSurface([61, 58, 59, 60], 26)
g.addStructuredSurface([62, 54, 63, 58], 27, marker=200)
g.addStructuredSurface([63, 3, 64, 59], 28)
g.addStructuredSurface([64, 60, 65, 57], 29, marker=100)
g.addStructuredSurface([62, 53, 65, 61], 30)

#Bottom right
g.addStructuredSurface([66, 67, 68, 69], 31)
g.addStructuredSurface([70, 66, 71, 55], 32, marker=200)
g.addStructuredSurface([71, 67, 72, 1], 33)
g.addStructuredSurface([72, 56, 73, 68], 34, marker=100)
g.addStructuredSurface([69, 73, 51, 70], 35)

#Add Volume for Web:
```

```
            #addStructuredVolume() takes three args. The first is a list of
            # surface IDs (structured surfaces).
            # The surfaces should make a hexahedron (i.e. 6 surfaces).
            # Other kinds of structured volumes than hexahedra will
            # not work for hexahedral elements, which is the only type of
            # 3D element that CALFEM handles.
            #The two optional parameters are the volume ID and volume marker.
            g.addStructuredVolume([0,1,2,3,4,5], 0, marker=90)

            #Add Volume for Top Flange:
            g.addStructuredVolume([1,6,7,8,9,10], 1) #top middle
            g.addStructuredVolume([11,12,13,14,15,10], 2) #top left
            g.addStructuredVolume([16,17,18,19,20,8], 3) #top right

            #Add Volume for Bottom Flange:
            g.addStructuredVolume([21,0,22,23,24,25], 4) #top middle
            g.addStructuredVolume([26,27,28,29,30,25], 5) #top left
            g.addStructuredVolume([31,32,33,34,35,23], 6) #top right

    elif self.cross_section_type=='C':
            wft=wft*2
            #Add Points for Web:
            g.addPoint([-tw/2, 0, 0], ID=0, marker=11)
            g.addPoint([tw/2, 0, 0], 1)
            g.addPoint([tw/2, L, 0], 2, marker=101)
            g.addPoint([-tw/2, L, 0], 3)
            g.addPoint([-tw/2, 0, hw], 4)
            g.addPoint([tw/2, 0, hw], 5, marker = 44)
            g.addPoint([tw/2, L, hw], 6)
            g.addPoint([-tw/2, L, hw], 7)

            #Add Points for Top Flange:
            #Top middle
            g.addPoint([-tw/2, 0, hw+tft], 8)
            g.addPoint([tw/2, 0, hw+tft], 9)
            g.addPoint([tw/2, L, hw+tft], 10)
            g.addPoint([-tw/2, L, hw+tft], 11)

            #Top right
            g.addPoint([wft/2, 0, hw], 16)
            g.addPoint([wft/2, 0, hw+tft], 17)
            g.addPoint([wft/2, L, hw+tft], 18)
            g.addPoint([wft/2, L, hw], 19)

            #Add Points for Bottom Flange:
            #Bottom middle
            g.addPoint([-tw/2, 0, -tft], 20)
            g.addPoint([tw/2, 0, -tft], 21)
            g.addPoint([tw/2, L, -tft], 22)
            g.addPoint([-tw/2, L, -tft], 23)

            #Bottom right
            g.addPoint([wft/2, 0, 0], 29)
            g.addPoint([wft/2, 0, -tft], 28)
            g.addPoint([wft/2, L, -tft], 31)
            g.addPoint([wft/2, L, 0], 30)


            #Add splines for Web:
            g.addSpline([0, 1], 0, elOnCurve = elOnCurve)
            g.addSpline([1, 2], 1, elOnCurve = elOnCurveLength)
            g.addSpline([2, 3], 2, elOnCurve = elOnCurve)
            g.addSpline([3, 0], 3, elOnCurve = elOnCurveLength, marker=12)
            g.addSpline([0, 4], 4, elOnCurve = elOnCurveWeb)
            g.addSpline([1, 5], 5, elOnCurve = elOnCurveWeb)
            g.addSpline([2, 6], 6, elOnCurve = elOnCurveWeb)
            g.addSpline([3, 7], 7, elOnCurve = elOnCurveWeb)
            g.addSpline([4, 5], 8, elOnCurve = elOnCurve)
            g.addSpline([5, 6], 9, elOnCurve = elOnCurveLength, marker=45)
            g.addSpline([6, 7], 10, elOnCurve = elOnCurve)
            g.addSpline([7, 4], 11, elOnCurve = elOnCurveLength)

            #Add splines for Top Flange:
            #Top middle
            g.addSpline([8, 9], 20, elOnCurve = elOnCurve)
            g.addSpline([9, 10], 21, elOnCurve = elOnCurveLength)
            g.addSpline([10, 11], 22, elOnCurve = elOnCurve)
            g.addSpline([11, 8], 23, elOnCurve = elOnCurveLength)
            g.addSpline([4, 8], 24, elOnCurve = elOnCurve)
            g.addSpline([5, 9], 25, elOnCurve = elOnCurve)
            g.addSpline([6, 10], 26, elOnCurve = elOnCurve)
            g.addSpline([7, 11], 27, elOnCurve = elOnCurve)

            #Top right
            g.addSpline([16, 17], 36, elOnCurve = elOnCurve)
            g.addSpline([17, 18], 37, elOnCurve = elOnCurveLength)
            g.addSpline([18, 19], 38, elOnCurve = elOnCurve)
            g.addSpline([19, 16], 39, elOnCurve = elOnCurveLength)
            g.addSpline([16, 5], 40, elOnCurve = elOnCurveFlange)
            g.addSpline([17, 9], 41, elOnCurve = elOnCurveFlange)
            g.addSpline([18, 10], 42, elOnCurve = elOnCurveFlange)
            g.addSpline([19, 6], 43, elOnCurve = elOnCurveFlange)

            #Add splines for Bottom Flange:
            #Bottom middle
            g.addSpline([20, 21], 50, elOnCurve = elOnCurve)
            g.addSpline([21, 22], 51, elOnCurve = elOnCurveLength)
            g.addSpline([22, 23], 52, elOnCurve = elOnCurve)
            g.addSpline([23, 20], 53, elOnCurve = elOnCurveLength)
            g.addSpline([0, 20], 54, elOnCurve = elOnCurve)
```

```
            g.addSpline([21, 1], 55, elOnCurve = elOnCurve)
            g.addSpline([22, 2], 56, elOnCurve = elOnCurve)
            g.addSpline([23, 3], 57, elOnCurve = elOnCurve)

            #Bottom right
            g.addSpline([28, 29], 66, elOnCurve = elOnCurve)
            g.addSpline([29, 30], 67, elOnCurve = elOnCurveLength)
            g.addSpline([30, 31], 68, elOnCurve = elOnCurve)
            g.addSpline([31, 28], 69, elOnCurve = elOnCurveLength)
            g.addSpline([21, 28], 70, elOnCurve = elOnCurveFlange)
            g.addSpline([1, 29], 71, elOnCurve = elOnCurveFlange)
            g.addSpline([2, 30], 72, elOnCurve = elOnCurveFlange)
            g.addSpline([22, 31], 73, elOnCurve = elOnCurveFlange)


            #Add surfaces for Web:
            g.addStructuredSurface([0, 1, 2, 3], 0)
            g.addStructuredSurface([8, 9, 10, 11], 1)
            g.addStructuredSurface([0, 4, 8, 5], 2, marker=200)
            g.addStructuredSurface([1, 5, 9, 6], 3)
            g.addStructuredSurface([2, 6, 10, 7], 4, marker=100)
            g.addStructuredSurface([3, 4, 11, 7], 5)

            #Add surfaces for Top Flange:
            #Top middle
            g.addStructuredSurface([20, 21, 22, 23], 6)
            g.addStructuredSurface([24, 20, 25, 8], 7, marker=200)
            g.addStructuredSurface([25, 21, 26, 9], 8)
            g.addStructuredSurface([26, 22, 27, 10], 9, marker=100)
            g.addStructuredSurface([27, 23, 24, 11], 10)

            #Top right
            g.addStructuredSurface([36, 37, 38, 39], 16)
            g.addStructuredSurface([40, 25, 41, 36], 17, marker=200)
            g.addStructuredSurface([41, 21, 42, 37], 18)
            g.addStructuredSurface([42, 38, 43, 26], 19, marker=100)
            g.addStructuredSurface([40, 9, 43, 39], 20)

            #Add surfaces for Bottom Flange:
            #Bottom middle
            g.addStructuredSurface([50, 51, 52, 53], 21)
            g.addStructuredSurface([0, 54, 50, 55], 22, marker=200)
            g.addStructuredSurface([55, 1, 56, 51], 23)
            g.addStructuredSurface([2, 57, 52, 56], 24, marker=100)
            g.addStructuredSurface([53, 54, 3, 57], 25)

            #Bottom right
            g.addStructuredSurface([66, 67, 68, 69], 31)
            g.addStructuredSurface([70, 66, 71, 55], 32, marker=200)
            g.addStructuredSurface([71, 67, 72, 1], 33)
            g.addStructuredSurface([72, 56, 73, 68], 34, marker=100)
            g.addStructuredSurface([69, 73, 51, 70], 35)

            #Add Volume for Web:
            #addStructuredVolume() takes three args. The first is a list of
            # surface IDs (structured surfaces).
            # The surfaces should make a hexahedron (i.e. 6 surfaces).
            # Other kinds of structured volumes than hexahedra will
            # not work for hexahedral elements, which is the only type of
            # 3D element that CALFEM handles.
            #The two optional parameters are the volume ID and volume marker.
            g.addStructuredVolume([0,1,2,3,4,5], 0, marker=90)

            #Add Volume for Top Flange:
            g.addStructuredVolume([1,6,7,8,9,10], 1) #top middle
            g.addStructuredVolume([16,17,18,19,20,8], 3) #top right

            #Add Volume for Bottom Flange:
            g.addStructuredVolume([21,0,22,23,24,25], 4) #top middle
            g.addStructuredVolume([31,32,33,34,35,23], 6) #top right

    elif self.cross_section_type=='Z':
            wft=wft*2
            #Add Points for Web:
            g.addPoint([-tw/2, 0, 0], ID=0, marker=11)
            g.addPoint([tw/2, 0, 0], 1)
            g.addPoint([tw/2, L, 0], 2, marker=101)
            g.addPoint([-tw/2, L, 0], 3)
            g.addPoint([-tw/2, 0, hw], 4)
            g.addPoint([tw/2, 0, hw], 5, marker = 44)
            g.addPoint([tw/2, L, hw], 6)
            g.addPoint([-tw/2, L, hw], 7)

            #Add Points for Top Flange:
            #Top middle
            g.addPoint([-tw/2, 0, hw+tft], 8)
            g.addPoint([tw/2, 0, hw+tft], 9)
            g.addPoint([tw/2, L, hw+tft], 10)
            g.addPoint([-tw/2, L, hw+tft], 11)

            #Top right
            g.addPoint([wft/2, 0, hw], 16)
            g.addPoint([wft/2, 0, hw+tft], 17)
            g.addPoint([wft/2, L, hw+tft], 18)
            g.addPoint([wft/2, L, hw], 19)

            #Add Points for Bottom Flange:
            #Bottom middle
            g.addPoint([-tw/2, 0, -tft], 20)
            g.addPoint([tw/2, 0, -tft], 21)
```

```
g.addPoint([tw/2, L, -tft], 22)
g.addPoint([-tw/2, L, -tft], 23)

#Bottom left
g.addPoint([-wft/2, 0, 0], 25)
g.addPoint([-wft/2, 0, -tft], 24)
g.addPoint([-wft/2, L, -tft], 27)
g.addPoint([-wft/2, L, 0], 26)

#Add splines for Web:
g.addSpline([0, 1], 0, elOnCurve = elOnCurve)
g.addSpline([1, 2], 1, elOnCurve = elOnCurveLength)
g.addSpline([2, 3], 2, elOnCurve = elOnCurve)
g.addSpline([3, 0], 3, elOnCurve = elOnCurveLength, marker=12)
g.addSpline([0, 4], 4, elOnCurve = elOnCurveWeb)
g.addSpline([1, 5], 5, elOnCurve = elOnCurveWeb)
g.addSpline([2, 6], 6, elOnCurve = elOnCurveWeb)
g.addSpline([3, 7], 7, elOnCurve = elOnCurveWeb)
g.addSpline([4, 5], 8, elOnCurve = elOnCurve)
g.addSpline([5, 6], 9, elOnCurve = elOnCurveLength, marker=45)
g.addSpline([6, 7], 10, elOnCurve = elOnCurve)
g.addSpline([7, 4], 11, elOnCurve = elOnCurveLength)

#Add splines for Top Flange:
#Top middle
g.addSpline([8, 9], 20, elOnCurve = elOnCurve)
g.addSpline([9, 10], 21, elOnCurve = elOnCurveLength)
g.addSpline([10, 11], 22, elOnCurve = elOnCurve)
g.addSpline([11, 8], 23, elOnCurve = elOnCurveLength)
g.addSpline([4, 8], 24, elOnCurve = elOnCurve)
g.addSpline([5, 9], 25, elOnCurve = elOnCurve)
g.addSpline([6, 10], 26, elOnCurve = elOnCurve)
g.addSpline([7, 11], 27, elOnCurve = elOnCurve)

#Top right
g.addSpline([16, 17], 36, elOnCurve = elOnCurve)
g.addSpline([17, 18], 37, elOnCurve = elOnCurveLength)
g.addSpline([18, 19], 38, elOnCurve = elOnCurve)
g.addSpline([19, 16], 39, elOnCurve = elOnCurveLength)
g.addSpline([16, 5], 40, elOnCurve = elOnCurveFlange)
g.addSpline([17, 9], 41, elOnCurve = elOnCurveFlange)
g.addSpline([18, 10], 42, elOnCurve = elOnCurveFlange)
g.addSpline([19, 6], 43, elOnCurve = elOnCurveFlange)

#Add splines for Bottom Flange:
#Bottom middle
g.addSpline([20, 21], 50, elOnCurve = elOnCurve)
g.addSpline([21, 22], 51, elOnCurve = elOnCurveLength)
g.addSpline([22, 23], 52, elOnCurve = elOnCurve)
g.addSpline([23, 20], 53, elOnCurve = elOnCurveLength)
g.addSpline([0, 20], 54, elOnCurve = elOnCurve)
g.addSpline([21, 1], 55, elOnCurve = elOnCurve)
g.addSpline([22, 2], 56, elOnCurve = elOnCurve)
g.addSpline([23, 3], 57, elOnCurve = elOnCurve)

#Bottom left
g.addSpline([24, 25], 58, elOnCurve = elOnCurve)
g.addSpline([25, 26], 59, elOnCurve = elOnCurveLength)
g.addSpline([26, 27], 60, elOnCurve = elOnCurve)
g.addSpline([27, 24], 61, elOnCurve = elOnCurveLength)
g.addSpline([24, 20], 62, elOnCurve = elOnCurveFlange)
g.addSpline([25, 0], 63, elOnCurve = elOnCurveFlange)
g.addSpline([26, 3], 64, elOnCurve = elOnCurveFlange)
g.addSpline([27, 23], 65, elOnCurve = elOnCurveFlange)

#Add surfaces for Web:
g.addStructuredSurface([0, 1, 2, 3], 0)
g.addStructuredSurface([8, 9, 10, 11], 1)
g.addStructuredSurface([0, 4, 8, 5], 2, marker=200)
g.addStructuredSurface([1, 5, 9, 6], 3)
g.addStructuredSurface([2, 6, 10, 7], 4, marker=100)
g.addStructuredSurface([3, 4, 11, 7], 5)

#Add surfaces for Top Flange:
#Top middle
g.addStructuredSurface([20, 21, 22, 23], 6)
g.addStructuredSurface([24, 20, 25, 8], 7, marker=200)
g.addStructuredSurface([25, 21, 26, 9], 8)
g.addStructuredSurface([26, 22, 27, 10], 9, marker=100)
g.addStructuredSurface([27, 23, 24, 11], 10)

#Top right
g.addStructuredSurface([36, 37, 38, 39], 16)
g.addStructuredSurface([40, 25, 41, 36], 17, marker=200)
g.addStructuredSurface([41, 21, 42, 37], 18)
g.addStructuredSurface([42, 38, 43, 26], 19, marker=100)
g.addStructuredSurface([40, 9, 43, 39], 20)

#Add surfaces for Bottom Flange:
#Bottom middle
g.addStructuredSurface([50, 51, 52, 53], 21)
g.addStructuredSurface([0, 54, 50, 55], 22, marker=200)
g.addStructuredSurface([55, 1, 56, 51], 23)
g.addStructuredSurface([2, 57, 52, 56], 24, marker=100)
g.addStructuredSurface([53, 54, 3, 57], 25)

#Bottom left
g.addStructuredSurface([61, 58, 59, 60], 26)
g.addStructuredSurface([62, 54, 63, 58], 27, marker=200)
g.addStructuredSurface([63, 3, 64, 59], 28)
```

```
                    g.addStructuredSurface([64, 60, 65, 57], 29, marker=100)
                    g.addStructuredSurface([62, 53, 65, 61], 30)

                    #Add Volume for Web:
                    #addStructuredVolume() takes three args. The first is a list of
                    # surface IDs (structured surfaces).
                    # The surfaces should make a hexahedron (i.e. 6 surfaces).
                    # Other kinds of structured volumes than hexahedra will
                    # not work for hexahedral elements, which is the only type of
                    # 3D element that CALFEM handles.
                    #The two optional parameters are the volume ID and volume marker.
                    g.addStructuredVolume([0,1,2,3,4,5], 0, marker=90)

                    #Add Volume for Top Flange:
                    g.addStructuredVolume([1,6,7,8,9,10], 1) #top middle
                    g.addStructuredVolume([16,17,18,19,20,8], 3) #top right

                    #Add Volume for Bottom Flange:
                    g.addStructuredVolume([21,0,22,23,24,25], 4) #bottom middle
                    g.addStructuredVolume([26,27,28,29,30,25], 5) #bottom left

                elType = 5 # Element type 5 is hexahedron.
                           # (See user manual for more element types)
                dofsPerNode= 3 #Degrees of freedom per node.

                mesher = GmshMesher(geoData = g,
                                    gmshExecPath = None,
                                    elType = elType,
                                    dofsPerNode= dofsPerNode)

                coords, edof, dofs, bdofs, _ = mesher.create()

                nDofs = size(dofs)
                ex, ey, ez = coordxtr(edof, coords, dofs)
                K = zeros([nDofs,nDofs])

                E = G/v
                ep = [2]
                D=hooke(4, E, v)

                for eltopo, elx, ely, elz in zip(edof, ex, ey, ez):
                    Ke = soli8ePortedFromCALFEM(elx, ely, elz, ep, D)
                    assem(eltopo, K, Ke)

                f = zeros([nDofs,1])

                bc = array([],'i')
                bcVal = array([],'i')

                if self.bc_type==0:
                        bc, bcVal = applybcPortedFromCALFEM(bdofs, bc, bcVal, 100, 0.0, 0)
                elif self.bc_type==1:
                        bc, bcVal = applybcPortedFromCALFEM(bdofs, bc, bcVal, 100, 0.0, 1)
                        bc, bcVal = applybcPortedFromCALFEM(bdofs, bc, bcVal, 101, 0.0, 2)
                        bc, bcVal = applybcPortedFromCALFEM(bdofs, bc, bcVal, 101, 0.0, 3)
                        bc, bcVal = applybcPortedFromCALFEM(bdofs, bc, bcVal, 200, 0.0, 2)
                elif self.bc_type==2:
                        bc, bcVal = applybcPortedFromCALFEM(bdofs, bc, bcVal, 100, 0.0, 0)
                        bc, bcVal = applybcPortedFromCALFEM(bdofs, bc, bcVal, 200, 0.0, 2)

                if self.load_type=='Point load':
                        applyforcePortedFromCALFEM(bdofs, f, 44, -P, 1)
                        applyforcePortedFromCALFEM(bdofs, f, 11, P, 1)
                elif self.load_type=='Distributed load':
                        applyforcePortedFromCALFEM(bdofs, f, 45, -P/(elOnCurveLength+1), 1)
                        applyforcePortedFromCALFEM(bdofs, f, 12, P/(elOnCurveLength+1), 1)

                a,r = solveq(K,f,bc,bcVal)

                ed = extractEldisp(edof,a)

                self.fEM.a=a
                self.fEM.coords=coords
                self.fEM.edof=edof
                self.fEM.dofsPerNode=dofsPerNode
                self.fEM.elType=elType


class St_Venant():
    def __init__(self,tau=0,rot=0):
        self.tau_max=tau
        self.tau_unit='Pa'
        self.rot_max=rot
        self.rot_unit=' '

class Vlasov():
    def __init__(self,tau=0,sigma=0,rot=0):
        self.tau_max=tau
        self.tau_unit='Pa'
        self.sigma_max=sigma
        self.sigma_unit='Pa'
        self.rot_max=rot
        self.rot_unit=' '

class FEM():
    def __init__(self,a=array([]),coords=array([]),edof=array([]),
                      dofsPerNode=0,elType=''):
        self.a=a
        self.coords=coords
        self.edof=edof
```

```
        self.dofsPerNode=dofsPerNode
        self.elType=elType
        self.mf=1
        self.undeformedMesh=False
        self.show_axis=True
        self.bgcolors=[[.9,.9,.9],[.3,.5,.7]]
        self.colorIndex=1
        self.viewProperties={'loc': (0.0, 0.4983273827238007, 0.0900000035762787),
                    'fov': 0.0, 'elevation': 30.0,
                        'zoom': 0.783688637065534,
                    'daspect': (1.0, 1.0, 1.0), 'azimuth': -10.0,
            'roll': 0.0}

class WorkThread(QtCore.QThread):
    def __init__(self):
        QtCore.QThread.__init__(self)

    def __del__(self):
        self.wait()

    def run(self):
        """Put computational code here. Send updates to GUI using the emit method."""
        for i in range(9):
            time.sleep(0.3)
            self.emit(QtCore.SIGNAL('update(QString)'), "from work thread " + str(i))

        self.emit(QtCore.SIGNAL("completed()"))
        return

if __name__=="__main__":

    vv_app.Create()
    app = Window()
    app.show()
    vv_app.Run()
```

Bilaga 3

Modul

```
import os, sys
sys.path.append(os.path.join(os.getcwd(),'..'))
import numpy as np
from numpy import *
import visvis as vv
from visvis.wibjects.colorWibjects import Colorbar
from visvis import Colormapable
from visvis.wobjects.textures import minmax

def soli8ePortedFromCALFEM(ex,ey,ez,ep,D,eq=None):
        """
        % PURPOSE
        %  Calculate the stiffness matrix for a 8 node (brick)
        %  isoparametric element.
        %
        % INPUT:   ex = [x1 x2 x3 ... x8]
        %          ey = [y1 y2 y3 ... y8]  element coordinates
        %          ez = [z1 z2 z3 ... z8]
        %
        %          ep = [ir]               ir integration rule
        %
        %          D                       constitutive matrix
        %
        %          eq = [bx; by; bz]       bx: body force in x direction
        %                                  by: body force in y direction
        %                                  bz: body force in z direction
        %
        % OUTPUT: Ke : element stiffness matrix
        %         fe : equivalent nodal forces
        %-----------------------------------------------------------------

        % LAST MODIFIED: M Ristinmaa   1995-10-25
        % Copyright (c)  Division of Structural Mechanics and
        %                Department of Solid Mechanics.
        %                Lund Institute of Technology
        %-----------------------------------------------------------------
        """

        ir=ep[0]
        ngp=ir*ir*ir
        #---------- gauss points ----------------------------------------
        if ir==1:
                g1=0.0
                w1=2.0
                gp=mat([g1,g1])
                w=mat([w1,w1])
        elif ir==2:
                g1=0.577350269189626
                w1=1.0
                gp = mat([
                        [-g1,-g1,-g1],
                        [ g1,-g1,-g1],
                        [ g1, g1,-g1],
                        [-g1, g1,-g1],
                        [-g1,-g1, g1],
                        [ g1,-g1, g1],
                        [ g1, g1, g1],
                        [-g1, g1, g1]
                ])
                w = mat([
                        [ w1, w1, w1],
                        [ w1, w1, w1],
                        [ w1, w1, w1],
                        [ w1, w1, w1],
                        [ w1, w1, w1],
                        [ w1, w1, w1],
                        [ w1, w1, w1],
                        [ w1, w1, w1]
                ])
        else:
                print "Used number of integration points not implemented"
                #return

        wp=multiply(multiply(w[:,0],w[:,1]),w[:,2])
        xsi=gp[:,0]
        eta=gp[:,1]
        zet=gp[:,2]
```

```
        r2=ngp*3

        #---------- shape functions ----------------------------------
        N=asmatrix(zeros((8,8)))
        N[:,0]=multiply(multiply(1-xsi,1-eta),1-zet)/8.
        N[:,4]=multiply(multiply(1-xsi,1-eta),1+zet)/8.
        N[:,1]=multiply(multiply(1+xsi,1-eta),1-zet)/8.
        N[:,5]=multiply(multiply(1+xsi,1-eta),1+zet)/8.
        N[:,2]=multiply(multiply(1+xsi,1+eta),1-zet)/8.
        N[:,6]=multiply(multiply(1+xsi,1+eta),1+zet)/8.
        N[:,3]=multiply(multiply(1-xsi,1+eta),1-zet)/8.
        N[:,7]=multiply(multiply(1-xsi,1+eta),1+zet)/8.

        dNr=asmatrix(zeros((r2,8)))
        dNr[0::3,0]=-multiply(1-eta,1-zet)
        dNr[0::3,1]= multiply(1-eta,1-zet)
        dNr[0::3,2]= multiply(1+eta,1-zet)
        dNr[0::3,3]=-multiply(1+eta,1-zet)
        dNr[0::3,4]=-multiply(1-eta,1+zet)
        dNr[0::3,5]= multiply(1-eta,1+zet)
        dNr[0::3,6]= multiply(1+eta,1+zet)
        dNr[0::3,7]=-multiply(1+eta,1+zet)
        dNr[1::3,0]=-multiply(1-xsi,1-zet)
        dNr[1::3,1]=-multiply(1+xsi,1-zet)
        dNr[1::3,2]= multiply(1+xsi,1-zet)
        dNr[1::3,3]= multiply(1-xsi,1-zet)
        dNr[1::3,4]=-multiply(1-xsi,1+zet)
        dNr[1::3,5]=-multiply(1+xsi,1+zet)
        dNr[1::3,6]= multiply(1+xsi,1+zet)
        dNr[1::3,7]= multiply(1-xsi,1+zet)
        dNr[2::3,0]=-multiply(1-xsi,1-eta)
        dNr[2::3,1]=-multiply(1+xsi,1-eta)
        dNr[2::3,2]=-multiply(1+xsi,1+eta)
        dNr[2::3,3]=-multiply(1-xsi,1+eta)
        dNr[2::3,4]= multiply(1-xsi,1-eta)
        dNr[2::3,5]= multiply(1+xsi,1-eta)
        dNr[2::3,6]= multiply(1+xsi,1+eta)
        dNr[2::3,7]= multiply(1-xsi,1+eta)
        dNr=dNr/8.


        Ke1=asmatrix(zeros((24,24)))
        fe=asmatrix(zeros([24,1]))
        JT=dNr*mat(vstack((ex,ey,ez))).T

        #---------- three dimensional case ---------------------------
        for i in range(ngp):
#               indx=mat([3*(i+1)-3, 3*(i+1)-2, 3*(i+1)-1 ])
                indx=[3*(i+1)-3, 3*(i+1)-2, 3*(i+1)-1 ]
                detJ=linalg.det(JT[indx,:])
                if detJ<10*finfo(float).eps:
                        print "Jacobideterminant equal or less than zero!"
                JTinv=linalg.inv(JT[indx,:])
                dNx=JTinv*dNr[indx,:]

                B=asmatrix(zeros([6,r2]))
                B[0,0::3]=dNx[0,:]
                B[1,1::3]=dNx[1,:]
                B[2,2::3]  =dNx[2,:]
                B[3,0::3]=dNx[1,:]
                B[3,1::3]=dNx[0,:]
                B[4,0::3]=dNx[2,:]
                B[4,2::3]  =dNx[0,:]
                B[5,1::3]=dNx[2,:]
                B[5,2::3]  =dNx[1,:]

                N2=asmatrix(zeros([3,r2]))
                N2[0,0::3]=N[i,:]
                N2[1,1::3]=N[i,:]
                N2[2,2::3]  =N[i,:]
                Ke1=Ke1+transpose(B)*D*B*asscalar(detJ)*asscalar(wp[i])

        if eq==None:
                return Ke1
        else:
                fe=fe+N2.T*eq*detJ*asscalar(wp[i])
                return Ke,fe
        #----------------------------end-------------------------------

def applyforcePortedFromCALFEM(boundaryDofs, f, marker, value=0.0, dimension=0):
    """
    Apply boundary condition to bcPresc and bcVal matrices.

    Parameters:

        boundaryDofs        Dictionary with boundary dofs.
        f                   force matrix.
        marker              Boundary marker to assign boundary condition.
        value               Value to assign boundary condition.
                            If not giben 0.0 is assigned.
        dimension           dimension to apply force. 0 - all, 1 - x, 2 - y, 3 - z

    """

    if boundaryDofs.has_key(marker):
        if dimension == 0:
            f[asarray(boundaryDofs[marker])-1] += value
        elif dimension == 1:
            f[asarray(boundaryDofs[marker][(dimension-1)::3])-1] += value
        elif dimension == 2:
```

```
            f[asarray(boundaryDofs[marker][(dimension-1)::3])-1] += value
        elif dimension == 3:
            f[asarray(boundaryDofs[marker][(dimension-1)::3])-1] += value
    else:
        print "Error: Boundary marker", marker, "does not exist."

def applybcPortedFromCALFEM(boundaryDofs, bcPrescr, bcVal, marker, value=0.0,

    """
    Apply boundary condition to bcPresc and bcVal matrices.

    Parameters:

        boundaryDofs        Dictionary with boundary dofs.
        bcPresc             1-dim integer array containing prescribed dofs.
        bcVal               1-dim float array containing prescribed values.
        marker              Boundary marker to assign boundary condition.
        value               Value to assign boundary condition.
                            If not giben 0.0 is assigned.
        dimension           dimension to apply bc. 0 - all, 1 - x, 2 - y

    """

    if boundaryDofs.has_key(marker):
        if (dimension==0):
            bcAdd = array(boundaryDofs[marker])
            bcAddVal = ones([size(bcAdd)])*value
        elif (dimension==1):
            bcAdd = boundaryDofs[marker][(dimension-1)::3]
            bcAddVal = ones([size(bcAdd)])*value
        elif (dimension==2):
            bcAdd = boundaryDofs[marker][(dimension-1)::3]
            bcAddVal = ones([size(bcAdd)])*value
        else:
            bcAdd = boundaryDofs[marker][(dimension-1)::3]
            bcAddVal = ones([size(bcAdd)])*value

        return hstack([bcPrescr,bcAdd]), hstack([bcVal,bcAddVal])
    else:
        print "Error: Boundary marker", marker, "does not exist."

def drawDisplacementsWithFillOption(displacements, coords, edof, dofsPerNode,
        elType, nodeVals=None, clim=None, axes=None,
        axesAdjust=True, doDrawUndisplacedMesh=True, magnfac=1.0,
        title=None, filled=False):
    '''
    Draws mesh with displacements in 2D or 3D. Scalar nodal values can also be
    drawn on the mesh. Returns the displaced Mesh object.
    Parameters:
    displacements-An N-by-1 array (or matrix). Row i contains the displacement of
                  dof i.
                  N-by-2 or N-by-3 arrays are also accepted, in which case row i
                  contains the x,y,z displacements of node i.
    coords      - An N-by-2 or N-by-3 array. Row i contains the x,y,z coordinates
                  of node i.
    edof        - An E-by-L array. Element topology. (E is the number of elements
                  and L is the number of dofs per element)
    dofsPerNode - Integer. Dofs per node.
    elType      - Integer. Element Type. See Gmsh manual for details. Usually 2
                  for triangles or 3 for quadrangles.
    nodeVals    - An N-by-1 array or a list of scalars. The Scalar values at the
                  nodes. nodeVals[i] should be the value of node i.
    clim        - 2-tuple. Colorbar limits (min, max). Defines the value range of
                  the colorbar. Defaults to None, in which case min/max are set
                  to min/max of nodeVals.
    axes        - Visvis Axes. The Axes where the model will be drawn.
                  If unspecified the current Axes will be used, or a new Axes will
                  be created if none exist.
    axesAdjust  - Boolean. True if the view should be changed to show the whole
                  model. Default True.
    doDrawMesh  - Boolean. True if mesh wire should be drawn. Default True.
    magnfac     - Float. Magnification factor. Displacements are multiplied by
                  this value. Use this to make small displacements more visible.
    title       - String. Changes title of the figure. Default None (in which case
                  title depends on other parameters).
    '''

    if displacements.shape[1] != coords.shape[1]:
        displacements = np.reshape(displacements, (-1, coords.shape[1]))
    displaced = np.asarray(coords + magnfac * displacements)

    if doDrawUndisplacedMesh:
        drawMesh(coords, edof, dofsPerNode, elType, axes, axesAdjust, title=title,
                         color=(0.5, 0.5, 0.5), filled=False)

    if nodeVals != None:
        m = drawNodalValues(nodeVals, displaced, edof, dofsPerNode, elType,
                         clim=clim, axes=axes,
                         axesAdjust=axesAdjust, doDrawMesh=True,
                         title=title)
    else:
        m = drawMesh(displaced, edof, dofsPerNode, elType, axes, axesAdjust,
                         title=title, filled=True)

    if title != None:
        vv.title(title, axes)

    return m
```