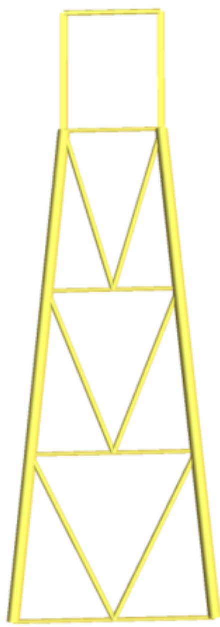
Kunnskap for en bedre verden

Analyse av en plan forskyvelig ramme med matrisemetoden

TMR4167 Marin Teknikk - Konstruksjoner

Kandidatnummer: 10032, 10068, 10089



November, 2021

Forord

Marin konstruksjonsteknikk innebærer dimensjonering og utforming av ulike konstruksjoner som skal operere ved marine miljø. I faget TMR4167 Marin Teknikk - konstruksjoner, har vi lært oss tilnærmingsmetoder for å kunne beregne deformasjon, rotasjon og momenter til konstruksjoner. I denne rapporten skal vi anvende matrisemetoden. Det er en metode som er godt egnet til når man skal regne bøyespenninger og deformasjoner på enkle konstruksjoner for hånd. I større system blir som regel antall ligninger og ukjente svært høyt slik at det er hensiktsmessig å bruke numeriske verktøy som setter ligningene i system.

Vi skal ved hjelp av Python ta inn en plan, forskyvelig ramme med påsatte laster som skal returnere tilhørende momenter, rotasjoner og bøyespenninger i konstruksjonen. Et velykket program skal være nesten like korrekt som et eksisterende dataprogram for rammeanalyse av marine konstruksjoner som brukes av ingeniørselskaper, men man har lært at det å stole blindt på dataprogram kan få fatale konsekvenser. Vi skal derfor kvalitetsteste dataprogammet med bruk av håndberegninger der vi bruker deformasjonsmetoden, samt DNV GL'S analyseprogram Nauticus 3D-Beam.

Prosjektet har gitt oss en dypere forståelse av matrisemetoden, samt hvordan vi kan bruke dataprogram for å analysere større system med flere ligninger. Arbeidsmengden har vært stor, og det har blitt brukt mye tid på feilsøking og testing for å få Python-programmet til å fungere så godt som mulig. Gruppen har hatt et godt samarbeid og en felles motivasjon for å ferdigstille en innholdsrik og god rapport.

Vi vil rette en stor takk til vår professor, Jørgen Amdahl som har lært oss den nødvendige kunnskapen for å kunne gjennomføre et slikt prosjekt.

Sammendrag

Denne rapporten er en fremstilling av resultatet av oppgaven gitt i emnet ”TMR4167 Marin Teknikk- Konstruksjoner”. Hensikten med oppgaven har vært å få økt forståelse for hvordan dataverktøy kan benyttes til daglige ingeniørberegninger samt oppbyggingen av disse.

Prosjektet har vært å programmere et slikt dataverktøy i Python som kan utføre en rammeanalyse på et todimensjonalt system. Programmet skulle bruke matrisemetoden som er en gunstig metode for store systemer der man vil regne på hele systemet som helhet, i motsetning til deformasjonsmetoden som tar for seg element for element. For å kontrollere at programmet regner riktig har vi benyttet programmet Nauticus®3D-Beam samt håndberegninger på enkle rammer.

Hensikten med oppgaven var å benytte seg av Python-programmet for å gjennom en iteresjonsprosess komme frem til tverrsnittsdata for konstruksjonen. Tverrsnittene skulle være dimensjonert slik at den mest belastede bjelken lå på 30-70% av flytspenningen. Selve jacket'en skulle bestå av av stål, og ha tre forskjellige rørtverrsnitt for henholdsvis bein, horisontalstag og diagonalstag. Dekkskontruksjonen skulle bestå av aluminium og ha to forskjellige I-profil for henholdsvis søyler og tverbjelke. Itereringen gjorde vi i 3D-beam siden vi brukte mye tid på å få programmet til å gå så bra som mulig, og bøyespenningene ikke blir korrekt regnet i vårt program.

Innhold

Figurer

Tabeller

1 Innledning

I denne rapporten blir det tatt for seg oppbygning og metoder bak et Python-program som skal være i stand til å analysere en plan forskyvelig ramme ved hjelp av matrisemetoden. For å kontrollere resultatene vil vi foreta håndberegninger på en enkel bjelke og en portalramme. Programmet 3D-Beam har også blitt brukt for å forenkle iterasjonsprosessen rundt dimensjoneringen.

1.1 Oppgaven

Oppgaven går ut på å lage et program i Python som gjør at man kan analysere en vilkårlig konstruksjon med matrisemetoden. For den gitte oppgaven er konstruksjonen vi skal analysere en dekkskonstruksjon som står på en jacket.

Konstruksjonen består av stål og aluminium for henholdsvis jacket og dekkskonstruksjon. Jacket'en består av rør med ulike diametere, mens dekkskonstruksjonen består av IPE bjelker. Det regnes samme flytspenning for begge material og kapasitetskrav for knekking antas å være oppfyllt uavhengig av de valgte tverrsnittsdimensjonene.

1.2 Konstruksjonsdata

Konstruksjonsdata - Verdier gitt av oppgaven	
Spenningselementer	**[MPa]**
E-Modul Stål	210 000
E-Modul Aluminium	70 000
Flytspenning σ_y	300
Lengder	**[m]**
L_1	25
L_2	18
L_3	18
Punktlaster	**[kN]**
P_1	4000
P_2	1000
Jevnt fordelte laster - proporsjonal med diameteren	**[kN/m]**
D = 1,5m: q_1	540
D = 1,5m: q_2	180

Table 1: Tabell over mål på konstruksjonen. For å se hvordan kreftene virker se Appendix 23

1.3 Forutsetninger og antagelser

Vi har gjort noen antagelser for oppgaven for å kunne benytte oss av superposisjonsprinsippet.

- Lineært elsastisk materiale
- Små deformasjoner
- Alle tverrsnitt bøyes om sin sterke akse
- Elementenes kapasitetskrav for knekking er oppfylt
- Egenvekten til konstruksjonen neglisjeres i beregningene
- Summen av alle moment i et knutepunkt er null

2 Fremgangsmåte

2.1 Matrisemetoden

Matrisemetoden er basert på deformasjonsmetoden, men i motsetning til deformasjonsmetoden tar matrisemetoden for seg systemet som helhet i stedet for ett og ett element. Dette gjør at matrisemetoden egner seg bedre for å løse store lineære ligningssystemer, gitt at man får litt regnehjelp av en datamaskin. I matrisemetoden etableres det en systemrelasjon bestående av den globale stivhetsmatrisen **K**, rotasjonsvektor **r** og lastvektor **R**

$$\mathbf{K}\mathbf{r} = \mathbf{R} \tag{1}$$

For mindre systemer vil det la seg gjøre å utføre håndberegninger med matrisemetoden, men for større system vil antall ligninger og ukjente bli tungvindt og krevende for hånd. En datamaskin derimot har ikke problemer med å utføre beregninger med flere ligninger og ukjente så lenge den får beskjed om hvordan den skal gå frem. Derfor er det hensiktsmessig å skrive et program som effektivt kan lese og organisere informasjon.

For å gå gjennom teoridelen av matrisemetoden benyttes kompendiet C.M Larsen, Kåre Syvertsen og Jørgen Amdahl 2010 [Larsen et al. 2010].

2.2 Diskretisering

Diskretisering går ut på å gjøre en oppdeling av konstruksjonen i et endelig antall elementer. Forbindelser mellom elementer kalles knutepunkter. Det er hensiktsmessig å la elementer og knutepunkter følge den naturlige oppbyggingen av konstruksjonen. Under vil man se hvordan vi har diskretisert vår ramme.

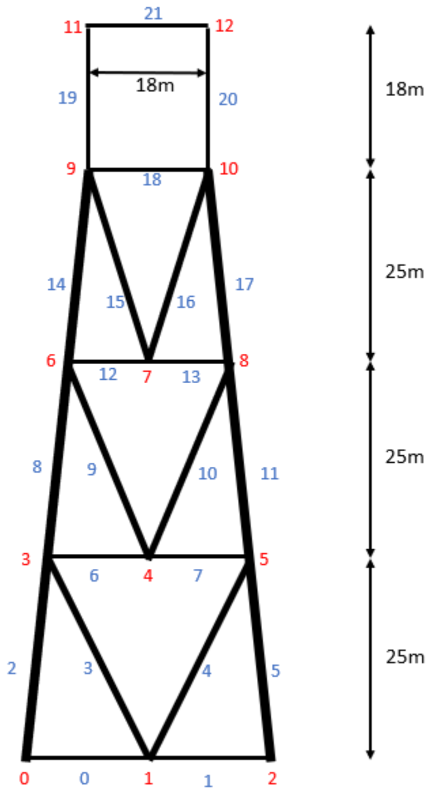


Figure 1: Diskretisering

Etter man har diskretisert konstruksjon kan det være fordelaktig å sette opp en sammenheng mellom de lokale og globale frihetsgradene i det som i elementmetoden kalles en konnektivitetsmatrise. Når man har satt opp en slik systematisk sammenheng er det et godt grunnlag for stivhetsmatrisen.

Element nr	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
Lokal Frihetsgrad 1	0	1	0	1	1	2	3	4	3	4	4	5	6	7	6	7	7	8	9	9	10	11
Lokal Frihetsgrad 2	1	2	3	3	5	5	4	5	6	6	8	8	7	8	9	9	10	10	10	11	12	12

Table 2: Konnektivitetsmatrise

2.3 Systemstivhetsmatrise

2.3.1 Lokal stivhetsmatrise

Før man kan sette opp systemstivhetsmatrisen må man ha stivhetsmatrisen til hvert enkelt element, den såkalte lokale stivhetsmatrisen. Metoden er lignende til deformasjonsmetoden, da stivhetsmatrisen til element i skrives på formen:

$$k_i = \begin{bmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{bmatrix} = \frac{E_i I_i}{L_1} \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix} \tag{2}$$

Der k_{11} og k_{22} er stivhetsleddene for ende 1 og 2, mens k_{12} og k_{21} er koblingsledd for moment i en ende og rotasjon i den andre. L_i er bjelkens lengde, E_i er bjelkens elastitetsmodul og I_i er bjelkens arealtreghetsmoment. Elementstivhetsmatrisen varierer kun med verdiene fra L, E og I. Ved 3 frihetsgrader per knutepunkt, vil dermed den lokale stivhetsmatrisen bli en 6x6-matrise.

2.3.2 Global stivhetsmatrise

Da den lokale stivhetsmatrisen kun gjelder per enkelt element vil de ikke tillate oss å regne på hele systemet som en helhet. Når man har funnet alle lokale stivhetsmatriser setter man opp systemstivhetsmatrisen, som også kalles den globale stivhetsmatrisen. Før dette er det også viktig å transformere den lokale stivhetsmatrisen til det globale koordinatsystemet.

Systemstivhetsmatrisen sin størrelse bestemmes av antall knutepunkt og frihetsgrader. I vårt tilfelle har konstruksjonen 13 knutepunkter med tre frihetsgrader hver som vil resultere i en 39x39 matrise. Hvert enkelt elementstivhetsmatrise adderes inn i systemstvhetsmatrisen ved hjelp av konnektivitetsmatrisen. Fordi slike systemstivhetsmatriser blir store og vanskelig å regne for hånd, er de gunstig for datamaskiner da man som nevnt kan regne på hele systemet som en helhet.

Under er det hentet et eksempel fra kapittel 8.7 i Kompendium for Marin Teknikk 2 som viser sammenhengen mellom elementstivhetsmatrise, konnektivitetsmatrise og systemstivhetsmatrise.

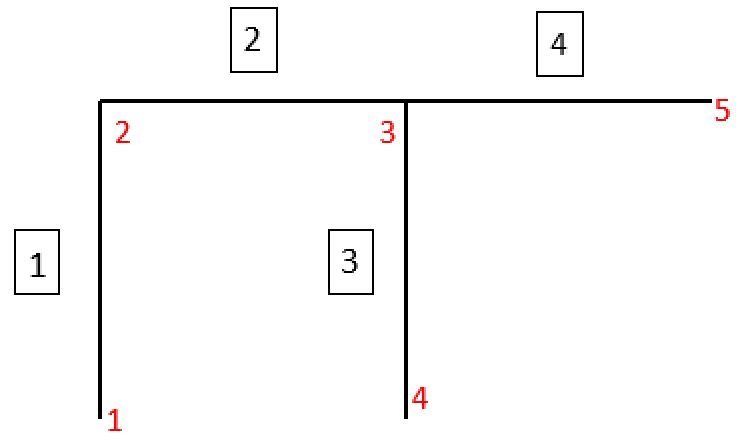


Figure 2: Eksempelramme

Etter man har diskretisert rammen kan man sette opp konnektivitetsmatrisen som er sammenhengen mellom de lokale og globale frihetsgradene. Ved hjelp av denne kan man sette opp systemstivhetsmatrisen.

Element nr.	**1**	**2**	**3**	**4**
Lokal Frihetsgrad 1	1	2	3	3
Lokal Frihetsgrad 2	2	3	4	5

Table 3: Konnektivitetsmatrise eksempelramme

For å sette opp systemstivhetsmatrisen adderes elementmatrisene inn i den globale stivhetsmatrisen **K**. Element (1,1) i den lokale stivhetsmatrisen til bjelkeelement 2, adderes til element (2,2) i den globale stivhetsmatrisen.

$$K = \begin{bmatrix} (k_{11})_1 & (k_{12})_1 & 0 & 0 & 0 \\ (k_{21})_1 & (k_{22})_1 + (k_{11})_2 & (k_{12})_2 & 0 & 0 \\ 0 & (k_{21})_2 & (k_{22})_2 + (k_{11})_3 + (k_{11})_4 & (k_{12})_3 & (k_{12})_4 \\ 0 & 0 & (k_{21})_3 & (k_{22})_3 & 0 \\ 0 & 0 & (k_{21})_4 & 0 & (k_{22})_4 \end{bmatrix}$$

2.4 Lastvektor

Etter å ha etbalert den globale stivhetsmatrisen mangler man bare lastvektoren R for å kunne løse systemrelasjonen

$$Kr = R. \tag{3}$$

Lastvektoren forteller hvilke ytre belastninger som påvirker konstruksjonen, og består av både konsentrerte knutepunktskrefter og momenter i knutepunktene, og knutepunktskrefter og moment som skyldes av laster i felt.

Dette finner vi ved å gå gjennom hver last, og regne ut fastinnspenningskreftene og momentene for hvert element og legge til dette med negativt fortegn. I tillegg summerer man inn punktkreftene som fungerer i knutepunktene direkte inn i lastvektoren. Det som er viktig er at fastinnspenningskreftene må transformeres til det globale koordinatsystemet før de legges inn i lastvektoren.

2.4.1 Fastinnspenningsmomenter

Fastinnspenningsmomenter er momenter forårsaket av ytre belastninger som virker på hver enkelt bjelke når alle rotasjonsfrihetsgradene er antatt fastholdt. Konsekvensen av at man ser på alle rotasjonsfrihetsgrader som fastholdt er at fastinnspenningsmomentene gir en momentfordeling som ikke tar hensyn til samvirke mellom elementene. Summen av fastinnspenningsmoment for elementer med felles knutepunkt kalles fastholdningsmomentet og angir momentet som kreves for å fastholde mot rotasjon i knutepunktet. For å beregne fastinnspenningsmoment benyttes tabell 8.3 gitt i kompendiet for fastinnspenningsmomenter under ulike lasttilfeller [Larsen et al. 2010]. I vårt tilfelle er det bare benyttet formlene for lineært fordelte laster.

Første tilfelle er om lasten er lineært stigende fra a til b, så vil en regne moment i henholdsvis ende a og b.

$$m_{ab} = -\frac{1}{30}pl^2 \qquad m_{ba} = \frac{1}{20}pl^2 \tag{4}$$

Andre tilfelle er lasten lineært avtagende fra a til b.

$$m_{ab} = -\frac{1}{20}pl^2 \qquad m_{ba} = \frac{1}{30}pl^2 \tag{5}$$

Etter beregning av fastinnspenningsmomenter vil man finne fastholdningsmomentet for hvert knutepunkt, for så finne bidraget for hvert knutepunkt til lastvektoren. Dette gjør man ved å legge sammen fastholdningsmoment og knutepunktslaster.

2.5 Randbetinglser

For å ta høyde for at noen av punktene er fast innspent må vi endre system stivhetsmatrisen. Randbetingelsene vi innfører er en veldig høy fjærstivhetskonstant. Denne legger vi til på diagonalen i stivhetsmatrisne for alle frihetsgradene i punkter som er fast innspent. Dermed vil forskyvningene og rotsasjonen gå mot null, desto høyere fjærstivhet man legger til.

2.6 Ligningsløsning

Etter å etablert systemrelasjonen og funnet både stivhetsmatrise og lastvektor, vil neste skritt være å finne de ukjente frihetsgradene. Regner man manuelt vil man på dette steget innføre noen randbetinglser for å stryke noen rader og kolonner i matrisen.

Siden vi skal regne på datamaskin har vi innført en rotasjonsfjær i alle knutepunkt langs diagonalen og satt den til 0 der det ikke er fast innspenning. I punktene der det er fastholding mot rotasjon innfører man en fjærstivhet på f.eks 10^6 ganger bøyestivheten.

For å finne kreftene i hvert element benyttes de lokale stivhetsmatrisene og lokale knutepunktsforskyvninger:

$$\mathbf{S_i} = \mathbf{k_i}\mathbf{v_i} + \mathbf{S} \tag{6}$$

der $\mathbf{k_i}\mathbf{v_i}$ er krefter fra forksyvning og $\mathbf{S}$ er fastinnspenningskrefter.[Larsen et al. 2010]

2.7 Bøyespenninger

Bøyespenningene er avgjørende for dimensjoneringen da det er utgangspunktet for å beregne flytspenningene. I oppgaven er det gitt at bøyespenningsnivået for den mest belastede bjelken skal ligge mellom 30%-70% av flytspenningen. For å beregne bøyespenningen bruker vi endemomentene og midtmomentene for hver bjelke.

$$\sigma = \frac{M}{I} \cdot z_{max} \tag{7}$$

σ er bøyespenningen, M er momentet og z er avstanden fra nøytralaksen i tverrsnittet til ytterte punkt.

2.8 Dimensjonering

For vår konstruksjon har vi tolket det som at jacket'en består av tre forskjellige rørprofil for henholdsvis bein horisontalstag og diagonalstag. Og at dekkskonstruksjonen vil ha to ulike I-profil for henholdsvis søyler og tverrbjelke. Jacket består av stål og dekkskonstruksjonen består av aluminium, men de har begge en flytspenning på 300MPa. Det er gitt i oppgaven at bøyespenningsnivået til den mest belastede bjelken skal ligge på 30%-70% av flytspenningen. For å dimensjonere dette riktig skal det gjennomføres en iterasjonsprossess i python-programmet og litt hjelp fra 3D-Beam.

2.9 3D-Beam

3D-Beam er en software utviklet av DNV GL som blir brukt for å modellere og analysere bjelkekonstruksjoner i 3D. I denne oppgaven har 3D-Beam blitt brukt for å analysere hovedkonstruksjonen og for å velge endelige tverrsnittsdimensjoner ved å gjennomføre iterasjoner. I tillegg har vi sammenlignet resultate fra håndberegninger og pythonprogrammet ved hjelp av 3D-beam. Ved å sammenligne moment, skjærkraft, bøyespenning og rotasjon kunne vi få en oversikt over hvor nøyaktig vår kode i python var.

3 Python

3.1 Python-funksjoner

Her kommer vi til å gå kort gjennom python-koden, noen av funksjonene forklarer vi i samme seksjoen, siden disse gjerne er underfunksjoner av hverandre. Koden tar utgangspunkt i koden utdelt fra NTNU 2020. Koden er også lagt ved i Appendix A.

3.1.1 main

Hovedfunksjonen i programmet, denne som kjører de andre funksjonene og lagrer eventuelle verdier for senere funksjoner.

3.1.2 structure_visualization

Funksjon for å plotte og visualisere rammen og deformasjonene. Denne koden fulgte med koden fra IMT, og vi har ikke gjort noe med denne.

3.1.3 Element (klasse)

Et objekt av denne klassen tilsvarer et element. Klassen består av variabler som er spesifikt koblet til elementet. Dimensjoner som kommer fra input filen, men også lokale krefter som regnes ut i programmet lagres i elementets klasse.

3.1.4 lesinput

Leser en textfil på formen beskrevet under Inputfil. Dataene lagres i ulike todimensjonale lister. Returnerer antall knutepunkt, knutepunktsdata, antall elementer, elementdata, antall laster, lastdata, antall profiler og profildata.

3.1.5 ant_elementer

Itererer gjennom alle elementene og lager et objekt av klassen Element for hvert bjelkeelement. Returnerer en liste med alle element-objektene i samme rekkefølge som elementnummereringen.

3.1.6 add_elements

Legger til dataen om hvert element i riktig element-objekt. Både dimensjoner fra inputfilen, men bruker også andre funksjoner som areal-funksjonen for å legge til dataen man trenger.

3.1.7 areal, ror_areal, ipe_areal

areal: Sjekker om profilet er ror- eller ipe-profil og bruker tilsvarende subrutine. Returnerer elementets areal
ror_areal og *ipe_areal*: Regner ut og returnerer tverrsnitts-arealet på profilet.

3.1.8 index_profildata

Tar inn valgt element og listen med profiler og itererer gjennom profilene for å matche elementets profiltype og riktig profil. Returnerer indexen til profilet i profil-listen.

3.1.9 andre_arealmoment, aam_ror_profil, aam_I_profil

andre_arealmoment: Tar inn lister med alle elementene og profilene, finner passende profildata ved å bruke en subrutine, index_profildata. Sjekker deretter om profilet er ipe- eller ror-profil og bruker tilsvarende subrutine. Returnerer en todimensjonal liste med andre arealmoment og avstand til arealsenter for hvert element.
aam_ror_profil og *aam_I_profil*: Regner ut andre arealmoment.

3.1.10 lengder

Regner ut lengden på alle elementene ved bruk av koordinatene til elementets punkter. Returnerer liste med alle lengdene.

3.1.11 global_rot

Bruker koordinatene på punktene til elementet for å regne på vinkelen til elementet i forhold til globalt koordinatsystem. Bruker så vinkelen til å lage 6x6-transformasjonsmatrisen til elementet og returnerer denne.

3.1.12 lokal_k

Tar inn elastisitetsmodul, areal, andre arealmoment of lengden til et elemet. Funksjonen regner ut, lager og returnerer elementes 6x6-stivhetsmatrise.

3.1.13 fastinnspenningskrefter

Tar inn listen med laster og listen med element-objekter. Funksjonen itererer gjennom alle lastene, sjekker om det er en punktlast eller fordelt last, kobler lasten til riktig element og regner ut fastinnspennings-moment og skjærkraft for begge endene. Legger så til listen med lokale fastinnspenningskrefter (fik) til elementets fik-variabel.

3.1.14 lastvektor_funk

Tar inn antall punkter og listen med element-objekter. Itererer gjennom alle elementene henter ut elementets transformasjonsmatrise og fastinnspenningsvektor og prikker disse for å få den globale fastinnspenningsvektoren. Legger så til denne med negativt fortegn inn i den globale lastvektoren, etter å ha regnet ut punktenes globale index. Returner den globale lastvektoren.

3.1.15 system_stivhetsmatrise, legg_til_K

system_stivhetsmatrise: Lager en kvadratisk matrise med tre ganger antall knutepunkter som størrelsen. Itererer gjennom elementene, og henter ut elementets globalt-roterte stivhetsmatrise. Bruker så subrutinen; *legg_til_K* for å legge hvert element på riktig plass i systemets stivhetsmatrise. Returnerer systemets stivhetsmatrise, K.

3.1.16 randbetingelser

Tar inn punkt-listen, system stivhetsmatrisen og lastvektoren og itererer gjennom alle punktene. Sjekker om punktet er fast innspent og adderer i så fall inn en fjærkraft på $10e8$ for gjøre deformasjonene tilnærmet lik 0. Setter også lastvektoren lik 0 i disse punktene. Returnerer deretter de oppdaterte versjonene av system stivhetsmatrisen og lastvektoren.

3.1.17 deformasjoner

Tar inn system stivhetsmatrisen og den globale lastvektoren, og prikker den inverse av system stivhetsmatrisen og den globale lastvektoren. Returnerer vektor med globale deformasjoner.

3.1.18 lokal_deform

Tar inn globale deformasjoner og listen med element-objekter. Itererer gjennom element-listen, og lager en vektor med deformasjonene for punktene til elementet. Transformerer de lokale deformasjonene til lokalt koordinatsystem

3.1.19 el_krefter

Itererer gjennom alle elementene henter ut elementenes lokale deformasjoner, stivhetsmatrise og fastinnspenningskrefter. Regner ut og lagrer endekreftene til elementet.

3.1.20 midt_moment

Itererer gjennom alle elementene og regner ut momentet på midten av bjelken basert på endekreftene.

3.1.21 maks_boyespenning

Itererer gjennom alle elementene og beregner boyespenningen ved begge endene og i midten av elementet. Lagrer så den største absoluttverdien av bøyespenningene til elementet.

3.1.22 print_funksjonene

I koden har vi laget 6 print funksjoner som printer for eksempel deformasjoner eller krefter på en fin og oversiktlig måte. Funksjonene: *print_deformajoner*, *print_knutepunktskrefter*, *print_opplager*, *print_M*, *print_Q* og *print_maks_boyespenning*.

3.2 Input fil

Programmet er lagt opp til at all data om konstruksjonen som skal analyseres skal leses inn gjennom én inputfil. Her følger en forklaring av oppbyggingen av selve filen og hva som blir tatt inn. Selve inputfilen ligger vedlagt i Appendix B.1

3.2.1 Knutepunkt

Den første verdien som leses inn er antall knutepunkt og blir lagret som en variabel. Påfølgende antall linjer vil samsvare med antall knutepunkt der man får oppgitt koordinat x og y relativt til knutepunkt 1. Tredje punkt forteller om knutepunktet er fastholdt mot rotasjon eller ikke, der verdien 0 tilsvarer fri rotasjon og verdien 1 er fastholding mot rotasjon. Hvert knutepunkt blir lagt til i en liste som et element.

3.2.2 Element

Veridien etter alle knutepunktene tilsvarer antall element og blir lagret i en variabel. Og for de antall element neste linjene vil verdiene bli lagret som et element i en liste over element. Et element leses inn som globalt knutepunkt 1, globalt knutepunkt 2, elastitetsmodul og profiltype. Vi har valgt å gi I-profiler tall som starter på 1, og hensikten bak dette er at man da kan gi inn så mange forskjellige I-profiler man vil. Rørprofiler vil på samme vis starte med tallet 2. Eksempelvis har vårt program to I-profil der de leses inn som henholdsvis 21 og 22. Dette gir programmet mulighet til å ta inn et stort utvalg innen de forskjellige profilene, men begrenser seg til 10 profiler.

3.2.3 Last

Etter innlesning av element vil antall laster bli lest inn og lagt i en variabel. Tilsvarende som for knutepunkt og element vil programmet da vite at de påfølgende linjene opptil antall laster er laster. Lastene leses inn som type last, hvilket element lasten virker på, størrelse på kraft, rotasjon og avstandskonstant. Det må presiseres at det er avholdt to plasser for innlesning av størrelse på last grunnet fordelte laster. For punktlaster vil man bare benytte seg av den første, og den andre vil settes lik 0. For fordelte laster vil man oppgi størrelse ved lokal ende 1 og størrelse ved lokal ende 2. Dette vil naturligvis begrense programmet til at det kun kan håndtere jevnt fordelte og lineært fordelte laster.

3.2.4 Profildata

Det leses inn antall forskjellige profiltyper og deretter legges de ulike typene i en liste. Profildata for I-pofil leses inn som profiltype, bredde på flens, tykkelse flens, høyde på steg og tykkelse for steg. For rør vil det være profildata, ytre radius og tykkelse på røret.

3.2.5 Flytspenning

Leser inn flytspenning for den endelige rammeanalysen. Vi antar én flytspenning for alle profiltyper og materialer, dette er en forenkling som fungerer for vår oppgave, men er lett å tilpasse om man trenger ulike flytspenninger.

4 Resultater

4.1 Kontroll av program

Det er viktig å kontrollere at Python-programmet fungerer slik vi har tenkt. For å sjekke at dette stemmer tok vi å sammenlignet resultatene fra programmet opp mot resultatene fra håndberegninger og programmet 3D-Beam. Grunnet forenklingene som ble gjort for håndberegningene ble areal satt til 10000 og elastitetsmodulen satt til 1. Dette var for å neglisjere aksialdeformasjon i programmet.

4.2 Testkonstruksjon 1

Testkonstruksjon 1 er en to-dimensjonal bjelke som er fast innspent i begge ender. Den har en jevnt fordelt last Q_1 som virker over hele bjelken, samt en punktlass P på midten.

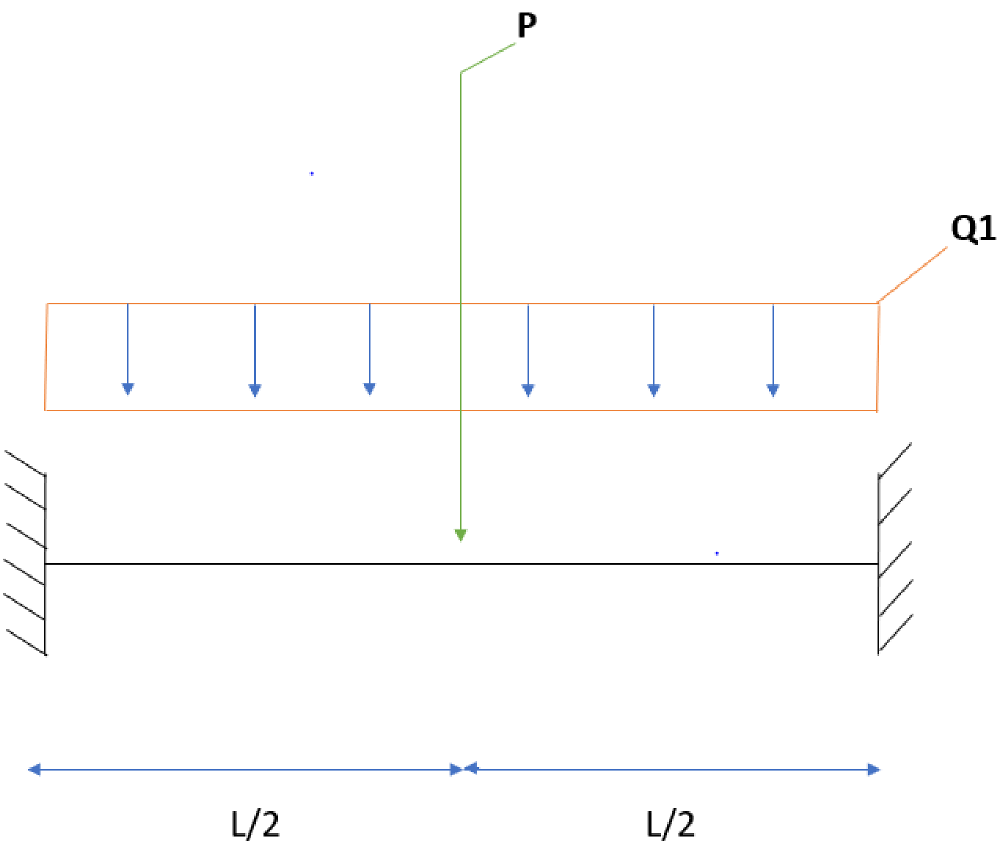


Figure 3: Testkonstruksjonen for den første håndberegningen

For å regne ut fastinnspenningsmoment tok vi i bruk kjente løsninger for moment og skjærkraft for henholdsvis en fast innspent bjelke med en punktlast på midten og en fast innspent bjelke med en jevnt fordelt last. Superposisjonsprinsippet ble så brukt for å regne ut momentdiagrammet for bjelken.

Vi så bort fra aksialdeformasjon ettersom bjelken ikke har noen horisontale krefter. Skjærkraft ble enkelt regnet ut ved å på ny ta i bruk kjente størrelser for de to ulike lasttilfellene. Hele beregningen og framgangsmåte er vist i appendix D.1 der vi har bestemt verdier; $P = 100kN$, $Q_1 = 10kN/m$ og $L = 4m$.

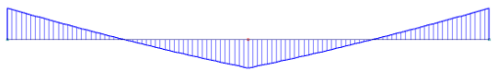


Figure 4: Momentdiagram: 3D-Beam

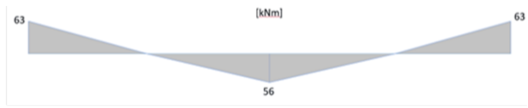


Figure 5: Momentdiagram: Håndberegninger

Figurene under viser to momentdiagram for systemet fra figur 3 med valgte verdier for lastene og lengden. Ett diagram er generert av 3D-Beam og det andre er tegnet for verdiene som håndberegningne ga oss. Vi samler alle verdiene opp i tabell 4 og ser da at 3D-beam, håndberegninger og Python-programmet stemmer overens.

Noder	**3D-Beam** [kNm]	**Python-program** [kNm]	**Håndberegninger** [kNm]
1	63	63	63
2	63	63	63

Table 4: Verdier for momentdiagram for bjelke [3]

4.2.1 Testkonstruksjon 2

Testkonstruksjon 2 er en to-dimensjonal fast innspent portalramme. Den har en horisontal punktlast P som virker i det venstre hjørnepunktet.

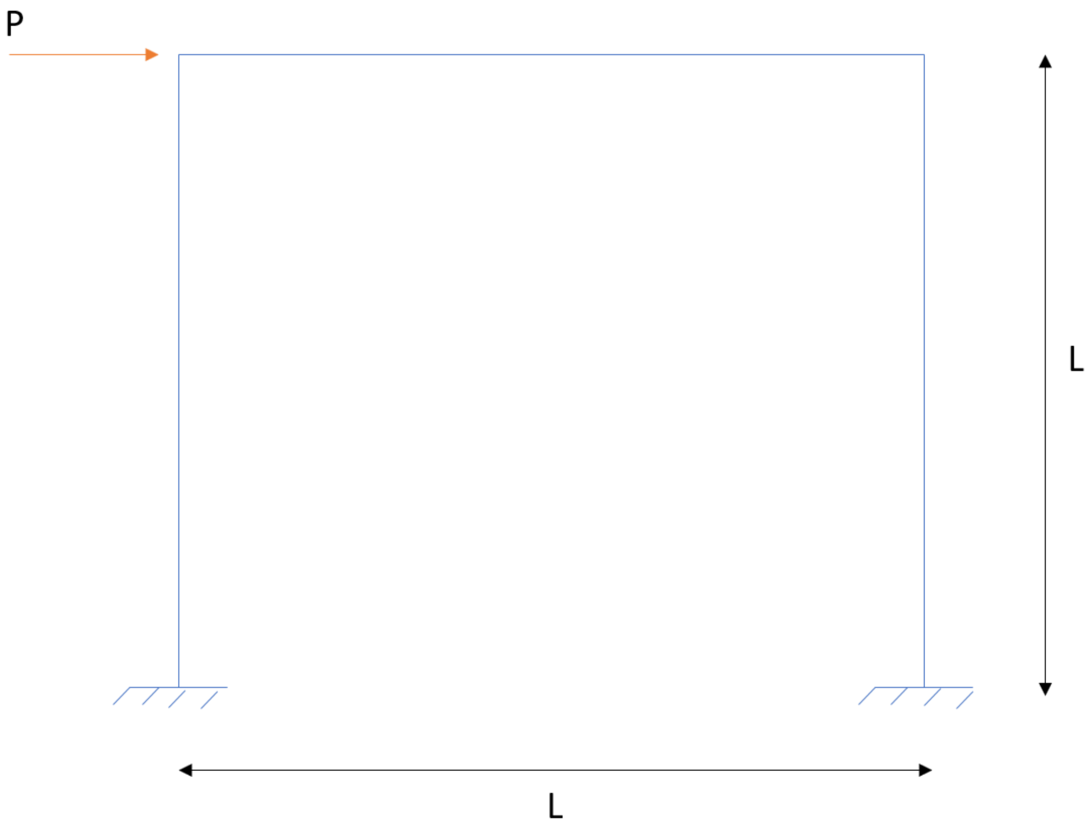


Figure 6: Testkonstruksjonen for den andre håndberegningen

For å regne på testkontruksjon 2 ble deformasjonsmetoden tatt i bruk. Det er en metode som egner seg godt til å løse problem der konstruksjonen kan deles inn i enkle element. For å gjøre håndberegningene litt enklere setter man noen randbetingelser slik at stivhetsmatrisen for elementene blir mer håndterlig. Fra oppgaven er det gitt at vi kan se bort fra aksialdeformasjoner og fra innspenningen i element 1 og 3 vil det verken være tverrforskyvning eller rotasjon. Videre antar vi stive hjørner som vil gi lik vinkelendring og at tverrforskyvningen er lik i begge ender. I utregningen er det benyttet en lengde på 4 meter og en punktlast på 1kN.

Bruk av superposisjon på momentdiagramene gir konstruksjonens totale momentdiagram. Beregning og fremgangsmåte er vist i appendix D.2

Figurene under viser to momentdiagram for portalrammen 6. Ett er generert av 3D-Beam og det andre er tegnet for verdiene som håndberegningene ga oss. Vi samler tallene moment, skjær- og aksialkraft i tabellene [5][6] og ser at de stemmer overens med hverandre.

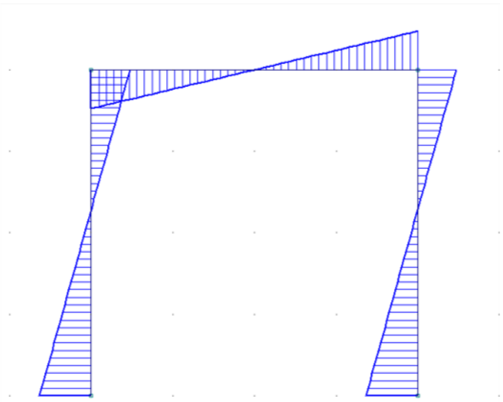


Figure 7: Momentdiagram: 3D-Beam

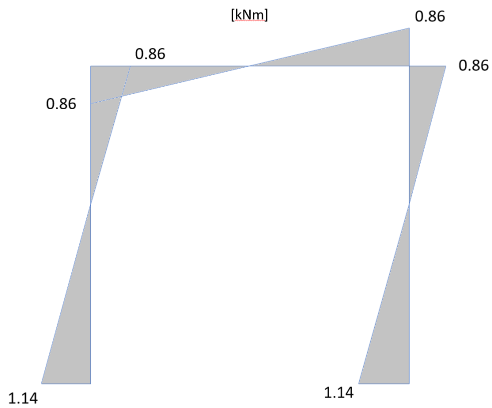


Figure 8: Momentdiagram: Håndberegninger

	Moment [kNm]					
	3D-beam		*Python*		*Beregning*	
Element	Ende 1	Ende 2	Ende 1	Ende 2	Ende 1	Ende 2
1	-1.14	0.86	-1.14	-0.86	-1.14	0.86
2	-0.86	0.86	0.86	0.86	-0.86	0.86
3	0.86	-1.14	-0.86	-1.14	0.86	-1.14

Table 5: Verdier for momentdiagram for portalramme [6]

	Skjærkraft [kN]					
	3D-beam		*Python*		*Beregning*	
Element	Ende 1	Ende 2	Ende 1	Ende 2	Ende 1	Ende 2
1	0.5	0.5	-0.5	-0.5	0.5	0.5
2	-0.43	-0.43	0.43	-0.43	-0.43	-0.43
3	-0.5	-0.5	-0.5	0.5	-0.5	-0.5

Table 6: Verdier for skjærkraft for portalramme [6]

	Aksialkraft [kN]					
	3D-beam		*Python*		*Beregning*	
Element	Ende 1	Ende 2	Ende 1	Ende 2	Ende 1	Ende 2
1	0.43	0.43	-0.43	0.43	0.43	0.43
2	-0.5	-0.5	0.5	-0.5	-0.5	-0.5
3	-0.43	-0.43	0.43	-0.43	-0.43	-0.43

Table 7: Verdier for aksialkraft for portalramme [6]

4.3 Valg av tverrsnittsprofil

Tverrsnittsdimensjonene skulle bestemmes slik at bøyespenningsnivået for den mest belastede bjelken i en gruppe av elementer var i størrelsesorden 30-70% av flytespenningen på 300 MPa. Vi delte konstruksjonen inn i fire grupper med elementer; ben (1), diagonalstag (2), tverrstag (3) og dekkskonstruksjon (4).

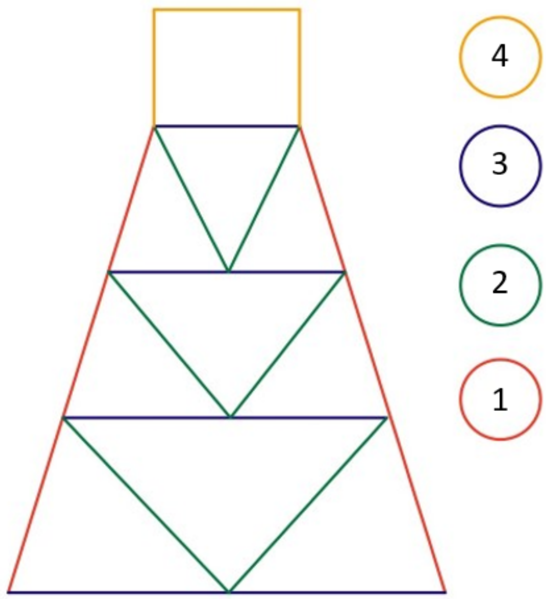


Figure 9: Gruppering av elementer for konstruksjonen

Dimensjonene ble først bestemt ut i fra hva vi trodde ville passe den gitte lastsituasjonen. Videre gjennomførte vi en iterasjonsprosess med antagelser om at kapasitetskravet for knekking var oppfylt uavhengig av de valgte tverrsnittsdimensjoner.

Vi tok i bruk 3D-beam for å lette bestemmelsen av dimensjonene. Etter å ha funnet dimensjoner som ga et passende forhold mellom bøyespenning og flytespenningen i hver gruppe, skulle vi bruke Python-programmet for den endelige analysen. Her fikk vi et stort avvik i sammenligning med svarene og besluttet dermed å gå for dimensjonene som ble iterert i 3D-beam. Dette blir diskutert videre i diskusjon 5.

Etter å ha gjennomført iterasjonsprossesen hadde vi bestemt oss for fem ulike tverrsnitt for konstruksjon. Jacket'en består av tre ulike rørprofiler (ben, diagonal- og horisontalstag) mens dekkskonstruksjonen består av to I-profiler (søyle og tverrbjelke). Dimensjonene for de ulike tverrsnittene er vist i tabell 8 og 9.

Dekkonstruksjon (I-bjelker)	Profil	Høyde i steg [mm]	Bredde flens [mm]	Tykkelse steg [mm]	Tykkelse flens [mm]
Søyler	I-bjelke	900	450	20	27
Tverrbjelker	I-bjelke	650	300	24	40

Table 8: Tverrsnittsdimensjoner for dekkonstruksjonen

Jacketen (rørprofil)	Ytre radius [mm]	Tykkelse [mm]
Bein	1500	150
Diagonal-stag	1000	250
Horisontal-stag	800	250

Table 9: Tverrsnittsdimensjoner for jacketen

Størst belastning er i element 17 med 43% av flytespenningen. Element 17 tilhører gruppen for beina til jacketen. Resten av gruppene har nå ved de bestemte tverrsnittene minst ett element som har bøyespenning mellom 30-70% prosent av flytespenningen. Elementene med størst belastning i hver gruppe er listet i tabell 10.

Gruppe	Profil	Element	Maksimal bøyespenning [MPa]	Prosent av flytespenning [%]
1	Rør	17	129	43%
2	Rør	16	93	31%
3	Rør	12	130	43%
4	I-bjelke	19	107	36%

Table 10: Den mest belasta bjelken i hver gruppe av elementer

4.4 Resultater fra Python-programmet

Vi har fått en rekke verdier fra Python-programmet og skal ved det gitte tversnittet fra tabellene 8 og 9 presentere moment, rotasjon, aksial- og skjærkrefter som blir gitt av henholdsvis koden og Nauticus 3D-Beam.

4.4.1 Moment

Tabellen 11 viser resultatene for moment fra Python-programmet sammenlignet med 3D-Beam. Tabellen viser moment ved start og endepunkt på bjelkene, i tillegg til midtmomentet for de bjelkene med fordelt last. Til slutt har vi regnet avviket mellom svarene fra koden og 3D-Beam.

	Ende 1 [kNm]			Ende 2 [kNm]			Midtmoment [kNm]		
Element	**3D**	**Python**	**Avvik**	**3D**	**Python**	**Avvik**	**3D**	**Python**	**Avvik**
0	-118	-94,8	19,7%	315	-267	15,2%			
1	-468	-421,8	9,9%	271	-249,6	7,9%			
2	4255	3985,8	6,3%	-398	57,6	85,5%			
3	388	341,2	12,1%	407	-507,7	24,7%			
4	395	347,5	12,0%	394	-495,6	25,8%			
5	4230	3958,4	6,4%	-396	50,9	87,1%			
6	-863	-909,5	5,4%	228	-151,2	33,7%			
7	-436	-362	17,0%	1065	-1115,3	4,7%			
8	872	1359,6	55,9%	13564	-13032,4	3,9%	117	-9446,8	7974%
9	303	226,4	25,3%	1874	-2084,3	11,2%			
10	361	286,8	20,6%	1756	-1962,8	11,8%			
11	1064	1560	46,6%	13188	-12639,5	4,2%	1064	-9077,6	753%
12	-7232	-8046,4	11,3%	9490	-10626,6	12,0%			
13	-9746	-10859,3	11,4%	7428	-8242,6	11,0%			
14	22670	23163	2,2%	-3901	3491,3	10,5%	-19022	25136	32%
15	9718	10859,3	11,7%	10574	-9374,2	11,3%	-11146	7347	34%
16	9518	10652,6	11,9%	10972	-9783,5	10,8%	-10935	6960	36%
17	22372	22844,9	2,1%	-2730	2276,8	16,6%	22372	23983	7%
18	7418	6667,8	10,1%	-7293	6539,7	10,3%			
19	746	-785	5,2%	-1868	-1867,7	0,0%			
20	949	967	1,9%	-1665	1685,7	1,2%			
21	-1868	1867,7	0,0%	-1665	-1685,7	1,2%			

Table 11: Sammenligning av momenter fra Python og 3D-beam

Vi har sammenlignet gjennomsnittsavviket til koden, ut i fra prosentavviket i tabell 11. Vårt høyeste avvik ligger på 87.1% noe som vil utgjøre en stor forskjell for gjennomsnittet. Vi ser også at tallene for midtmoment ikke stemmer overens og har derfor kun regnet gjennomsnittsavvik for ende 1 og ende 2. Dette blir diskutert videre i diskusjon.

	Ende 1	**Ende 2**	**Totalt**
Gjennomsnittsfeil	13.86%	18.22	16.04%

Table 12: Gjennomsnittsverdier for avviket fra tabell 11 for momentverdier

Vi tegnet momentdiagramet til konstruksjonen ut i fra momentverdiene fra python-programmet.

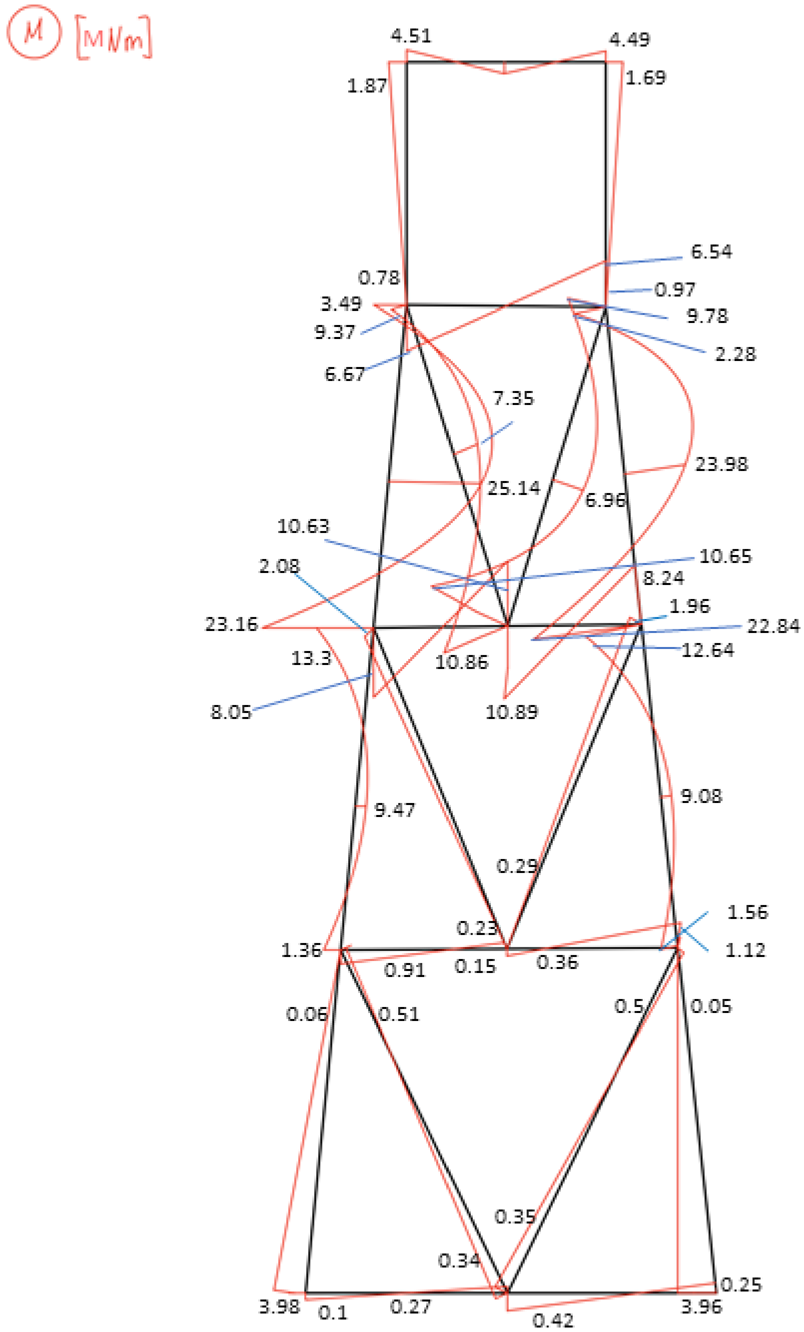


Figure 10: Momentdiagram basert på Pythonresultat

Momentdiagrammet fra 3D-beam viser null moment i dekkonstruksjonen, 11, noe som er feil. Dette er siden vi har rotert bjelke-elementene på dekkskonstruksjonen slik at deformasjonene vil skje over tverrsnittets sterke akse.

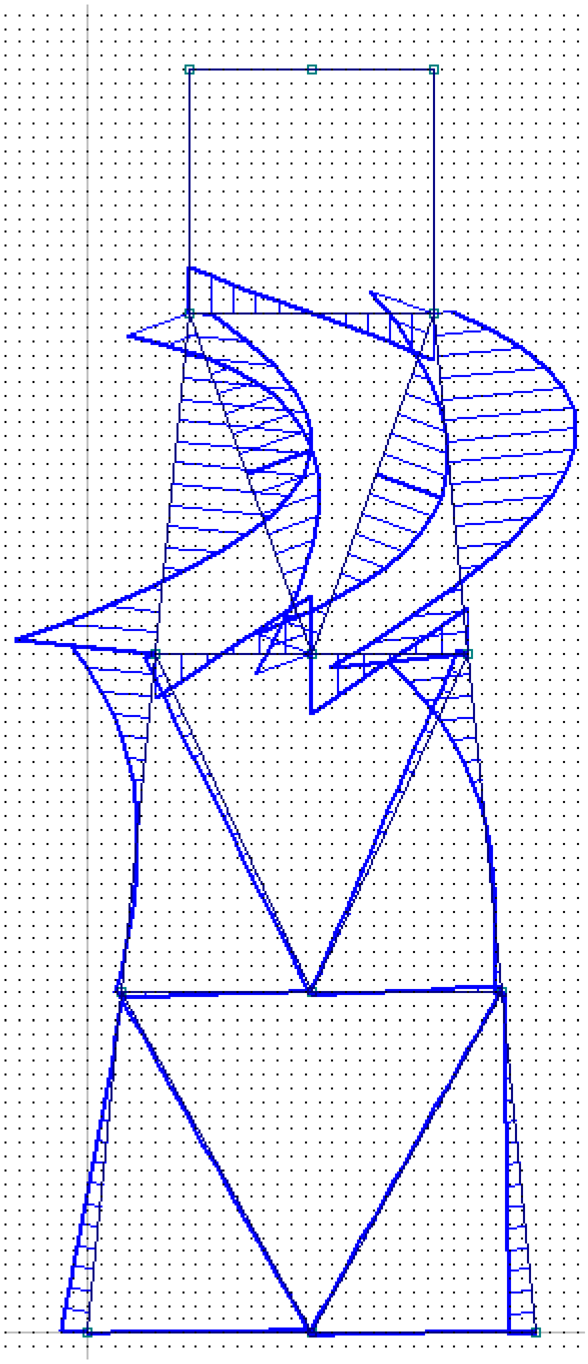


Figure 11: Momentdiagram fra 3D-Beam

4.4.2 Skjærkraft

Vi har sammenlignet resultatene fra skjærkraft i tabell 13 og regnet på avviket mellom koden og 3D-Beam.

	Ende 1 [kN]			Ende 2 [kN]		
Element	**3D**	**Python**	**Avvik**	**3D**	**Python**	**Avvik**
0	-26,23	21,9	16,53%	-26,2	-21,9	16,53%
1	-44,8	40,7	9,21%	-44,8	-40,7	9,21%
2	185,2	-160,9	13,13%	185	160,9	13,13%
3	-0,657	5,8	782,80%	-0,657	-5,8	782,80%
4	0,033	5,2	15657%	0,033	-5,2	15657%
5	184,1	-159,6	13,31%	184,1	159,6	13,31%
6	-77,9	75,8	2,73%	-77,9	-75,8	2,73%
7	-107,2	105,5	1,60%	-107,2	-105,5	1,60%
8	249	-1042,9	318,84%	-2013	-2738,8	36,06%
9	-57	67,5	18,42%	-57	-67,5	18,42%
10	-51	60,9	19,41%	-51	-60,9	19,41%
11	271	-1066,5	293,54%	-1990	-1194,7	39,96%
12	-1454	1623,7	11,67%	-1454	-1623,7	11,67%
13	-1493	1663,3	11,41%	-1493	1663,3	11,41%
14	4826	-6337,1	31,31%	4826	2707,8	43,89%
15	2625	-3775,8	43,84%	-3752	2601,2	30,67%
16	2602	3752,6	44,22%	-3775	2624,4	30,48%
17	4768	-6276,1	31,63%	-21489	2768,8	87,12%
18	817	-733,8	10,18%	817	733,8	10,18%
19	145	147,4	1,66%	145	-147,4	1,66%
20	145	-147,4	1,66%	145	147,4	1,66%
21	-511	-510,1	0,18%	489	-489,9	0,18%

Table 13: Sammenligning av skjærkrefter fra Python og 3D-beam

Tabell 13 viser gjennomsnittsverdier for avviket for skjærkraft. Vårt høyeste avvik er i element 4 på 15657%. Vi har sitt bort i fra element 4 i gjennomsnittsberegningen, ettersom det er en liten verdi og vil bare bidra til et mer unøyaktig resultat. Dette blir diskutert videre i diskusjon.

,

	Ende 1	**Ende 2**	**Totalt**
Gjennomsnittsfeil	79.87%	56.29%	68.07%

Table 14: Gjennomsnittsverdier for avviket fra tabell 13 for skjærkraft

Vi tegnet diagrammet til skjærkraft ut i fra momentverdiene fra python-programmet og sammenlignet det med diagrammet fra 3D-Beam. Diagrammene vises i figur 12 og 13.

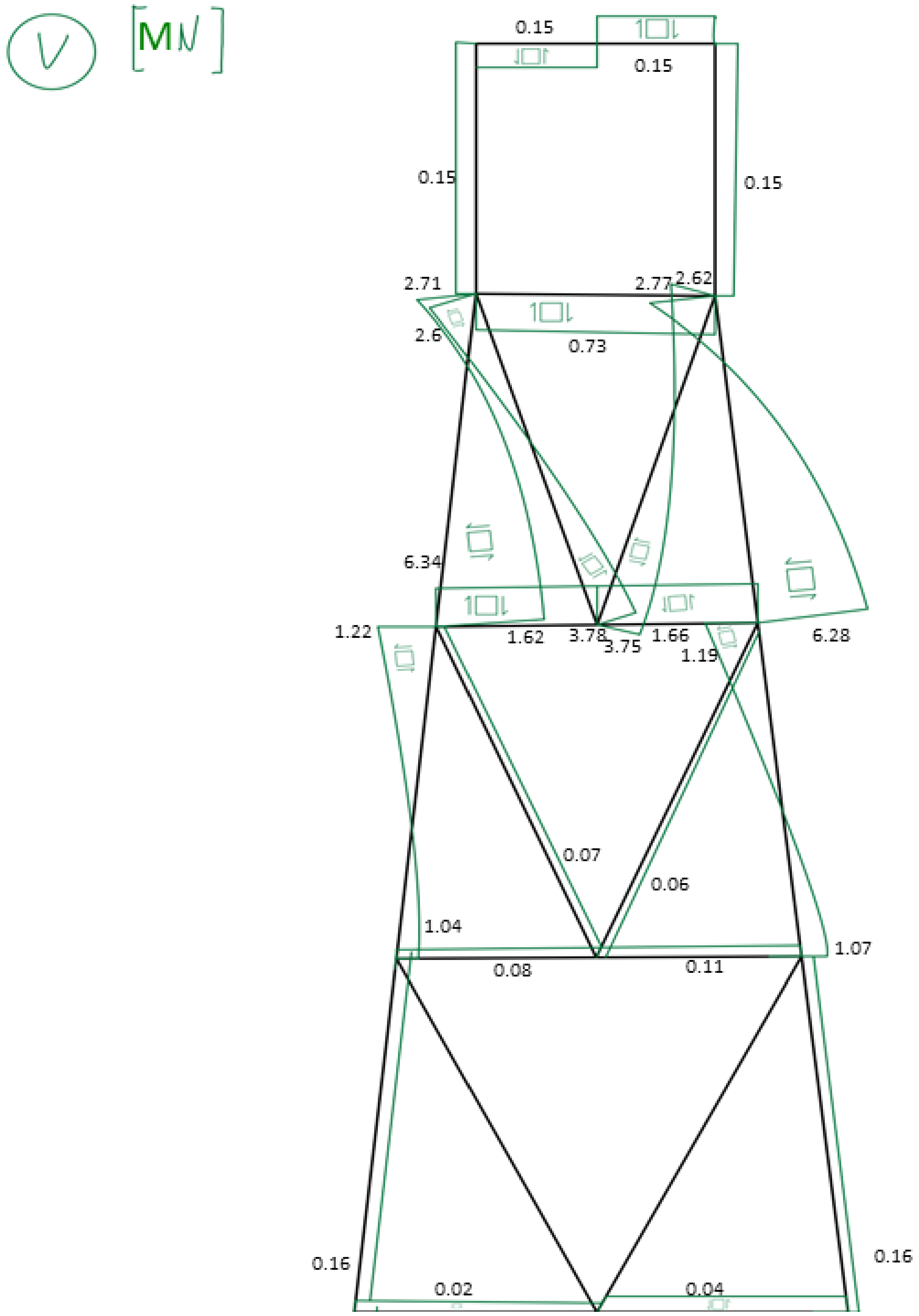


Figure 12: Skjærdiagram basert på Pythonresultat

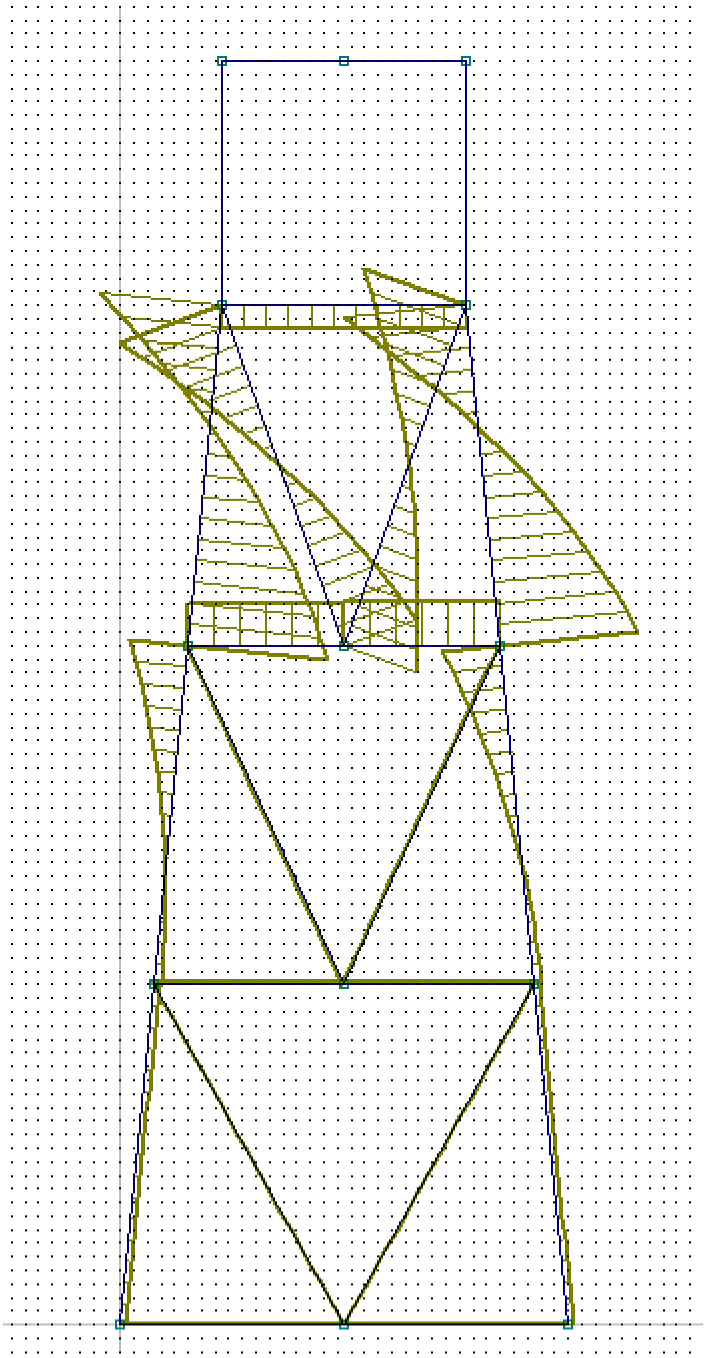


Figure 13: Skjærkraft fra 3D-Beam

4.4.3 Aksialkraft

Python-programmet returnerte alle verdiene for skjærkraft ved bjelkeendene, slik at vi sammenligne resultatene fra 3D-Beam som vises i tabell 15.

	Aksialkrefter [kN]		
Element	**3D**	**Python**	**Avvik**
0	10656	11194	5,05%
1	-10656	-11194	5,05%
2	59909	54746	8,62%
3	-21820	-22930	5,09%
4	21799	22909	5,09%
5	-68936	12321	82,13%
6	12509	-12287	1,77%
7	-12476	34648	177,72%
8	40704	-29597	27,29%
9	-30025	29567	1,53%
10	29996	-43665	45,57%
11	-49722	7568	84,78%
12	8498	-7648	10,00%
13	-8574	6720	21,62%
14	12489	-12030	3,68%
15	-17967	11980	33,32%
16	17917	-15719	12,27%
17	-21489	15719	26,85%
18	-254	250	1,57%
19	-4511	4510	0,02%
20	-4489	4489	0,00%
21	-145	-147	1,38%

Table 15: Sammenligning av aksialkrefter fra Python og 3D-beam

Tabell 15 viser avvikene i aksialkraft for de ulike elementene som til slutt ble regnet til en gjennomsnittsverdi på 25.47%.

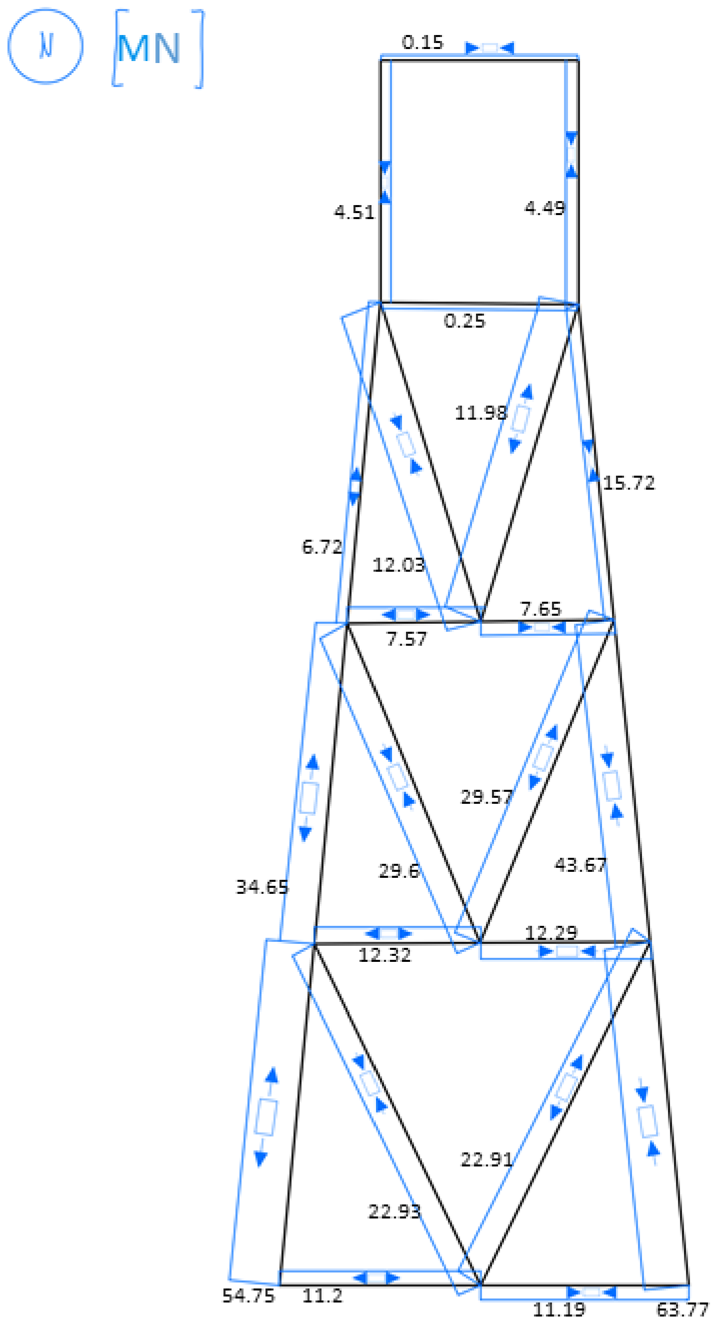


Figure 14: Aksialkraftdiagram basert på Pythonresultat

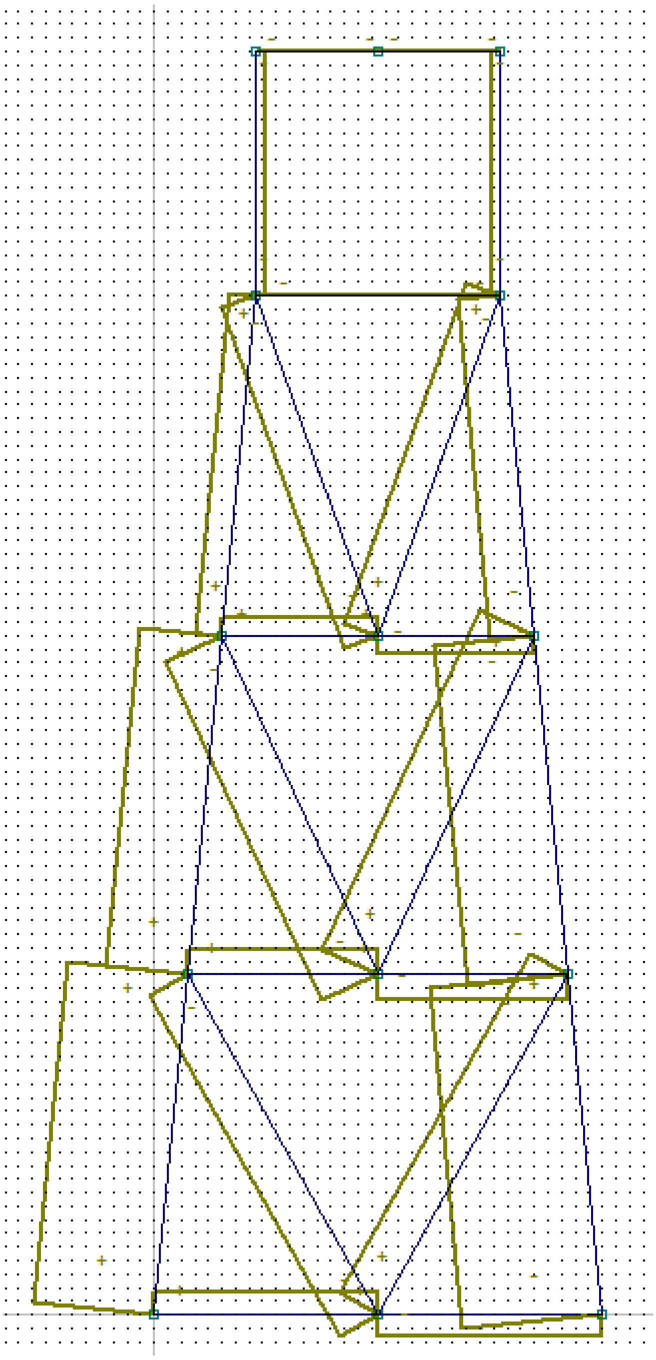


Figure 15: Aksialkraftdiagram fra 3D-Beam

4.4.4 Forskyvninger og rotasjoner

Vi regnet ut konstruksjonens forskyvning i x-retning, y-retninger og rotasjoner i Python-programmet. Vi sammenlignet verdiene med 3D-Beam, og fikk avvik fra 3.18%-12.76%.

	X-forskyvning [mm]			Y-Forskyvning [mm]			Rotasjon [rad]		
Node	**3D**	**Python**	**Avvik**	**3D**	**Python**	**Avvik**	**3D**	**Python**	**Avvik**
0	0,00	0,00	0%	0,00	0,00	0%	0	0%	0%
1	1,94	2,04	5,05%	0,85	0,85	0,01%	0,00039	0,00034	12,71%
2	0,00	0,00	0%	0,00	0,00	0,00%	0,00000	0%	0,00%
3	28,86	27,91	3,29%	8,44	7,56	10,44%	0,00157	0,00160	1,82%
4	30,79	29,82	3,18%	1,70	1,70	0,00%	0,00157	0,00032	36,05%
5	28,87	27,92	3,29%	10,14	9,26	8,69%	0,00156	0,00159	1,93%
6	75,43	70,65	6,33%	11,47	9,83	14,32%	0,00360	0,00361	0,30%
7	76,50	71,61	6,40%	2,56	2,56	0,01%	0,00673	0,00719	6,83%
8	75,42	70,64	6,33%	14,89	13,24	11,04%	0,00351	0,00352	0,23%
9	120,83	107,07	11,39%	9,29	7,46	19,74%	0,00419	0,00381	9,04%
10	120,78	107,02	11,39%	14,41	12,58	12,73%	0,00392	0,00353	9,84%
11	84,80	74,12	12,60%	18,13	19,96	10,08%	0,01823	0,01781	2,29%
12	83,86	73,16	12,76%	41,70	39,87	4,38%	0,01823	0,01789	1,87%

Table 16: Sammenligning av forskyvninger og rotasjoner fra Python og 3D-beam

	X-retning	Y-retning	Radianer	Totale gjennomsnitt
Gjennomsnittsfeil	7.45%	8.31%	7.53%	7.76%

Table 17: Gjennomsnittsverdier for avviket fra tabell 16 for forskyvninger og rotasjoner

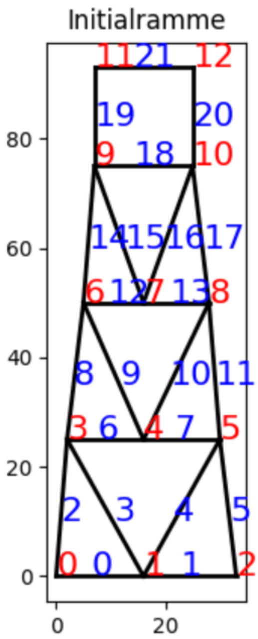


Figure 16: ramme før deformasjon

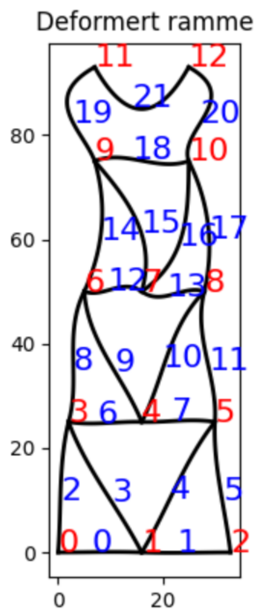


Figure 17: ramme deformert

5 Diskusjon

For å sjekke at programmet fungerer har vi brukt håndberegningene som sammenligningsgrunnlag og sjekket disse i 3D-beam også. For begge håndberegningene fikk python-programmet vårt korrekte svar målt opp mot både 3D-beam og håndberegningene selv. I (Table 4) ser vi at alle utregningene gir samme endemoment på 63kNm. For portal-rammen ble også tallverdiene for alle momentene (Table 5), skjærkreftene (Table 6) og aksialkreftene (Table 7) helt like i python, 3D-Beam og for håndberegningene. Men vi ser også i tabellene at fortegnene noen steder ble forskjellig. Det har med måten positiv ende blir definert, selv om vi trodde dette skulle gå fint, tror vi nå det kan være et større problem enn først antatt.

Da vi kjørte programmet for hele rammen vår fikk vi likevel store avvik, som har ført til mye tidsbruk for å feilsøke koden. De største avvikene mellom Python-programmet og 3D-Beam kommer riktignok for de minste verdiene, se ende 2 på de nederste beina(element 2 og 5) i Table 11. Her er avviket på over 85% grunnet altfor lave verdier på moment i Python-programmet. Ellers når vi sammenligner elementenes momentkrefter i 3D-Beam med eget program, ser vi at momentfortegnet på spesielt ende 2 ofte er motsatt av resultatene fra 3D-Beam. Vi har prøvd å rotere de lokale koordinatsystemene til noen av bjelkene i 3D-Beam, likevel så er ikke fortegnet konsekvent forskjellig. Dette gir oss grunn til å tro at vi har noe fortegnsfeil, eller lignende, i koden. Denne fortegnsfeilen var hovedgrunnen til at vi har slitt med å regne midtmomentet korrekt, da vi ikke vet helt sikkert hvilket fortegn de ulike endene får. Midt momentet er dermed ikke korrekt regnet ut, og vi antar at vi har feil i funksjonenes utregninger i tillegg til eventuelle følgefeil.

Generelt er det likevel en gjennomgående sammenheng mellom resultatene, og vi ser i Table **??** at det gjennomsnittlige moment-avviket ligger på rundt 16%.

Avviket øker dermed betraktelig når vi ser på både aksialkreftene (Table 15) og spesielt skjærkreftene (Table 13). Blant annet ser vi i ende 2 på element 4 at 3D-Beam gir en skjærkraft på 0.033kN, mens python programmet vår gir -5,2 kN. Så store forskjeller gir nok en bekreftelse på at vi har noe utregningsfeil i koden. Prosentavviket til disse verdiene kan også vise kunstig høye tall da de gjelder for de minste verdiene i systemet og dermed vil utgjøre relativ liten forskjell.

Siden det er størst feil i aksial- og skjær-kreftene kan det kanskje tyde på at feilen ligger i noe av transformeringene mellom lokalt og globalt koordinatsystem. Eventuelt så kan det ha noe med de aksielle deformasjonene å gjøre siden håndberegningene ga riktige resultater, og der neglisjerte vi de aksielle deformasjonenene ved å sette opp arealet.

Det er derimot på deformasjonene og rotasjonene programmet vårt får analysens minste gjennomsnittsavvik, se Table 16. Størst deformasjonsavvik får vi i det 9'ende knutepunktets y-retinng med 19.74%, men ellers er er de fleste avvikene på under 10% som betrygger oss om at programmet i hvert fall fungerer til en viss grad. I Figure 22 vises rammen oppskaler deformert form.

For å finne tverrsnittet tok vi i bruk 3D-Beam for å lette bestemmelser av dimensjonene. Tanken var å først bruke 3D-Beam til å finne dimensjoner som ga et passende forhold mellom bøyespenning og flytespenning i hver gruppe, før vi skulle bruke Python-programmet for den endelige analysen. Her fikk vi større problemer enn først antatt, da sammenligningen for bøyespenningsverdiene ga store avvik. Etter mye feilsøking på koden måtte vi til slutt ta et valg om hvilke data vi skulle gå etter. Valget falt på 3D-Beam siden vi så på det som et tryggere valg, da den gav oss mer realistiske verdier for konstruksjonen og oppfylte kravene til oppgava.

Etter bestemmelse av tverrsnittsdimensjoner fikk vi prosenter av flytespenningen som var helt nede i 31% som er på grensa til oppgavens krav på 30-70%. Vi tror mulig at avvikene vi regnet ut i [resultater 4] ville ha blitt redusert om vi hadde minsket dimensjonene på tverrsnittene. Det hadde også vært helt mulig ettersom vi skulle se bort i fra knekking og hadde mye å gå på ut i fra oppgavens krav om belastning på bjelkene.

6 Konklusjon

I dette prosjektet har vi fått jobbet med programmeringsferdighetene våre og skapt en Python-kode som kan ta inn en vilkårlig todimensjonal konstruksjon og analysere konstruksjonen med både punktlaster og lineært fordelte laster. Programmet vi står igjen med etter de siste ukenes harde arbeid, er dessverre ikke helt korrekt. Vi ser likevel klare paralleller mellom resultatene fra python-programmet vårt og resultatene fra 3D-Beam. For den avgjørende analysen av dekkskonstruksjonen får vi et gjennomsnittlig avvik på moment på rundt 16%, og til tross for relativt lave avvik på mange av skjærkreftene hadde noen ender spesielt store avvik på over 15000%. Når det kommer til forskyvninger og rotasjoner er ikke gjennomsnittsavviket på mer enn rundt 7%.

Selv med utallige timer debugging/feilsøking av koden har vi ikke funnet feilen, og vi har mange teorier om hvor feilen kan ligge, fra fortegn og transformering av koodinatsystemener til at det har med aksialdeformasjonene. Boyespenningsutregningen var også så langt fra realistisk at vi måtte gå for all itterering i 3D-beam.

Vi kan likevel ta med oss erfaringer fra bruk av digitale verktøy som brukes daglig til ingeniørberegninger og noe av teorien bak disse programmene. Det å forsøke å lage et slikt program og ikke lykkes har vært en lekse i hvorfor man ikke alltid kan stole på digitale verktøy uten noen form for forståelse bak programmene. I tillegg har vi lært hvor nyttig og effektive disse programmene er når man kan vurdere og kontrollere til en viss grad resultater er fornuftig.

Referanseliste

Larsen, Carl Martin, Jørgen Amdahl and Kåre Syvertsen (2010). *Kompendium Marin teknikk 2.* norsk. NTNU.

NTNU, Department of Marine Technology (2020). *IMT Software Wiki - LaTeX.* URL: https://www.ntnu.no/wiki/display/imtsoftware/LaTeX (visited on 15th Sept. 2020).

Appendix

A Python-kode

A.1 main.py

```
 import numpy as np

from structure_visualization    import *
from AA_les_input               import *
from AA_klas                    import *
from AB_lengder                 import *
from AC_dimensjoner_last        import *
from AB_areal_og_I              import *
from AD_fim                     import *
from AE_lokal_k                 import *
from AE_transformasjonsmatrise  import *
from AE_lastvektor              import *
from AE_store_stivhetsmatrisen  import *
from AF_randbetingelser         import *
from AG_endemomenter            import *
from AP_print                   import *
from AG_midtmoment              import *
from AH_boyespenning            import *
from AF_deformasjoner           import *





# -----Rammeanalyse-----
def main():
    # -----Initialiserer figurer-----
    fig_init, ax_init, fig_def, ax_def = setup_plots()
    # -----Til visualiseringen, velg f rste indeks brukt i nummerering av noder og element-----
    first_index = 0


    # -----Leser input-data-----
    n_punkter, punkter, n_elementer, elementer, n_laster, laster_udimensjonert, n_profiler, profiler, flytspenning = lesinput()

    elem_list = ant_elementer(n_elementer)
    add_elements(n_elementer, elementer, elem_list, profiler, punkter)



    # -----Plott initalramme-----
    plot_structure(ax_init, punkter, elementer, 1, first_index)

    # Dimensjonerer de fordelte lastene til diameteren p   r rprofilene de virker p
    laster = dimensjoner_fordeltlast(profiler, laster_udimensjonert, elementer)
    #laster =laster_udimensjonert








    # -----Fastinnspenningsmomentene------
    # Regner ut fastinnspenningsmomentene for elementene
    #fim, fiq = fastinnspennings_mq( n_elementer,elem_list, laster, elem_list)
    fastinnspenningskrefter(elem_list,  laster)
    # -----Lastvektor-----
    lastvektor = lastvektor_funk(n_punkter,laster,  elem_list)

    # -----Systemets stivhetsmatrise-----
    K_system = system_stivhetsmatrise(elem_list, n_punkter,punkter)
```

```
    # -----Justerer stivhetsmatrisen og lastvektoren til randbetingelsene -----
    K, L_vek = randbetingelser(n_punkter,punkter,K_system,lastvektor)


    # -----Sjekk av deformasjoner -----
    deform = deformasjoner(K, L_vek)

    # -----Deler deformasjonsvektoren i rotasjoner, x- og y-forskyvning -----
    rot = deform[2::3]
    x_def = deform[0::3]
    y_def = deform[1::3]


    # -----Regner ut knutepunktkreftene -----
    lokal_deform(elem_list, deform)
    el_krefter(elem_list)
    midt_moment(elem_list)
    mom(elem_list, laster)
    maks_boyespenning(elem_list)
    prosent_av_flyt(flytspenning, elem_list)


    #
    print_deformasjoner(rot, x_def, y_def, n_punkter)
    print_knutepunktskrefter(elem_list)
    print_opplager(elem_list, punkter)
    print_M(elem_list)
    print_Q(elem_list)
    print_maks_boyespenning(elem_list, flytspenning)

    # -----Plott deformert ramme-----
    skalering = 1e-2  # Du kan endre denne konstanten for   skalere de synlige deformasjonene til rammen
    plot_structure_def(ax_def, punkter, elementer, 1, first_index, -skalering * rot)
    plt.show()

main()
```

A.2 class Element

```
class Element(object):
    '''
    En klasse for hvert bjelke-element med variabler for   holde oversikt over alle st rrelser koblet til hvert lokale element
    '''
    # Index p  de lokale endene
    ende_1 = int
    ende_2 = int

    # Punktdataen til de lokale endene; x-/ y-koordinater og fast innspent eller ikke
    punkt_1 = []
    punkt_2 = []

    # Material- og tverrsnittsdata:
    #Elastisitetsmodul
    E = int
    #Andre arealtreghetsmoment
    I = float
    #Areal
    A = float
    #Elementlengde
    L = float
    #Avstand til tverrsnittets arealsenter
    z_c = float

    #Profiltype: Tosifret tall der f rste tall er valg av profil (1_ = I-profil, 2_ = ror-profil)
```

```
    #            og andre siffer er ulike varianter/tykkelser for profilet. EKSEMPEL: 13 = I-profil nr 3
    profil_type = int
    #Profildata/tverrsnittsm l til elementet
    profil = []

    #Lagrer kreftene p  en fordelt last som virker p  elementet. Tar bare h yde for  n fordelt last per element
    fordeltlast = []

    #Lokal stivhetsmatrise
    k_lokal = np.zeros((6,6))
    #Lokal stivhetsmatrise rotert til globalt koordinatsystem
    k_global = np.zeros((6, 6))

    #Transformasjonsmatrise
    T = np.zeros((6, 6))

    #Lokale fast innspenningskrefter
    fik  = np.zeros((6,1))
    #Lokal lastvektor
    lokal_lastvektor = np.zeros((6,1))

    # Lokale deformasjoner og rotasjoer
    deformasjoner = np.zeros((6,1))

    #Endekrefter
    krefter = np.zeros((6,1))

    #Moment og skjerkraft midt p  bjelken
    M_midt = float
    Q_midt = float

    #St rste absolutte b yespenning av spenningen ved endene eller midten
    sigma_max = float
    #Prosentvise st rrelsen p  b yespenningen basert p  valgt flytspenning
    prosent_av_flyt = float
```

A.3 ant_elementer.py

```
def ant_elementer (n_elem):
    '''
    Itererer gjennom alle elementene, oppretter et Element-objekt for hver og legger til i en liste.
    :param n_elem: antallet elementer
    :return: liste med alle elementene som objekter av Element-klassen
    '''

    elem_list = []
    for i in range(n_elem):
        # Oppretter et objekt av klassen: Element
        element  = Element()
        # Legger til objektet i listen: elem_list
        elem_list.append(element)
    return elem_list
```

A.4 add_elements

```
def add_elements (n_elem, elementer, elem_list, profiler, punkter):
    '''
    Legger til inputdataen som f.eks. dimensjoner til hvert Element-objekt. I tillegg kj res subrutiner for
       legge til blant annet areal og lokal stivhetsmatrise.
    :param n_elem: Antall elementer
    :param elementer: Liste med alle elementene slik de leses inn fra inputfilen
    :param elem_list: Liste med Element-objekter av klassen Element
    :param profiler: Liste med ulike profiler slik de leses inn fra inputfilen
```

```
    :param punkter: Liste med alle punktene og deres koordinater, slik lest inn fra inputfilen
    '''

    # Liste med andre arealmoment for hvert element sortert i samme rekkef lge som elementene
    I = andre_arealmoment(n_elem, elementer, profiler)

    # Liste over alle elementlengdene, sortert i samme rekkef lge
    elem_lengder = lengder(punkter, elementer, n_elem)

    for i in range(n_elem):
        # Elementdataen fra inputfilen
        elem = elementer[i]
        # Element-objektet der dataen skal lagres
        e = elem_list[i]

        #Lagrer punkt-indeksen til de lokale endene
        e.ende_1     = elem[0]
        e.ende_2     = elem[1]
        # Lagrer punktdataen ( x, y, fastinnsp.) til de lokale endene
        e.punkt_1 = punkter[e.ende_1]
        e.punkt_2 = punkter[e.ende_2]

        # Lagrer henholdsvis: elementstivhetn, andrearealmoment, elementlengden og z-avstanden til tverrsnittsarealet
        e.E          = elem[2]
        e.I          = I[i][0]
        e.L          = elem_lengder[i]
        e.z_c        = I[i][1]

        # Lagrer elementets profiltype (tosifret)
        e.profil_type = elem[3]

        # Finner indexen til riktig profil og lagrer til profilets profildata
        index = index_profildata(elem, profiler)
        p_data = profiler[index]
        e.profil = p_data


        # Regner ut og lagrer profilets areal
        A = areal(elem, p_data)
        e.A = A
        # Areal og andre arealmoment brukt for   kontrollere programmet opp mot h ndberegningene
        # Arealet er s pass h yt for   neglisjere aksiale deformasjoner
        e.A = 100000
        e.I = 1

        # Lagrer transformasjonsmatrisen (6x6-matrise)
        T = transformasjon(elementer[i], punkter, e.L)
        e.T = T

        # Lagrer den lokale stivhetsmatrisen
        e.k_lokal = lokal_k(elem, e.A, e.I, e.L)

        # Regner den inverse av transformasjonsmatrisen
        T_inv = np.linalg.inv(T)

        # Bruker transformasjonsmatrisen og den inverse til   regne ut den
        # lokale stivhetsmatrise for det globale koordinatsystemet
        e.k_global = np.linalg.multi_dot([T, e.k_lokal, T_inv])
```

A.5 les_input.py

```
def lesinput():
    #  pner inputfilen
    file_in_data = open("input.txt", "r")

    # Leser totalt antall punkter og lagrer antallet som int
    n_punkter = int(file_in_data.readline())
```

```

    # Leser alle punktene og lagrer i en liste der hvert punkt best r av x-
    koordinat, y-koordinat og fastinnspenning
    punkter = np.loadtxt(file_in_data, dtype=float, max_rows=n_punkter)


    # Leser antall elementer
    n_elem = int(file_in_data.readline())

    # Leser inn element-data; lokalt punkt 1, lokalt punkt 2, Elastisitetsmodul,
    Profiltype
    # Elementnummer tilsvarer radnummer i "elem"-variabel
    elementer = np.loadtxt(file_in_data, dtype=int, max_rows=n_elem)


    # Leser antall laster som virker p  rammen
    n_last = int(file_in_data.readline())

    # Leser lastdata;
    laster = np.loadtxt(file_in_data, dtype=float, max_rows=n_last)

    # Leser antall profiltyper totalt, b de IPE og ror
    n_profiler = int(file_in_data.readline())

    # Leser profildata;
    profiler = np.loadtxt(file_in_data, dtype=float, max_rows=n_profiler)

    # Leser flytspenning, antar  n flytspenning for alle tverrsnitt
    flytspenning = int(file_in_data.readline())

    # Lukker input-filen
    file_in_data.close()

    return n_punkter, punkter, n_elem, elementer, n_last, laster,n_profiler,
    profiler, flytspenning
```

A.6 areal.py

```
 def areal(element, profildata):
    '''
    Sjekker hvilket profil det er og bruker subrutiner til   regne ut arealet
    :param element: Element
    :param profildata: Tverrsnittsdata til valgt element
    :return: tverrsnittsareal
    '''

    type_profil = element[3]

    # Sjekker om profilet er IPE
    if str(type_profil)[:1] == '1':
        A = ipe_areal(profildata)

    # Sjekker om profilet er r r
    elif str(type_profil)[:1] == '2':
        A = ror_areal(profildata)
    return A


def ror_areal (profildata):
    '''
    :param profildata: Tverrsnittsdata for et r r-profil
    :return: Arealet til r r-tverrsnittet
    '''
    #Ytre radius
    R = profildata[1]
    #R rtykkelsen
    t = profildata[2]

    A = ((np.pi * R**2) - (np.pi * (R - t)**2))
    return A

```

```
def ipe_areal (profildata):
    '''
    :param profildata: Tverrsnittsdata for et I-profil
    :return: Arealet til et I-tverrsnitt
    '''
    #Arealet p  flensene
    a_flens = profildata[1] * profildata[2]
    #Arealet p  steget
    a_steg  = profildata[3] * profildata[4]

    A       = 2 * a_flens + a_steg
    return A
```

A.7 andre_arealmoment.py

```
 def andre_arealmoment (n_elem, elementer, profildata):
    '''
        Regner ut andre arealmoment for alle elementene og lagrer de i en liste
        :param n_elem: Antall elementer
        :param elementer: Liste over elementene (fra inputfilen)
        :param profildata: Liste over m l  p  profiler (fra inputfilen)
        :return: Todimensjonal liste med andre arealmoment og arealsenter for hvert element
        '''

    # Lager listen som skal returneres med alle arealmomentene
    I = np.zeros((n_elem, 2))

    #Itererer gjennom hvert element hvor    regne for hvert
    for i in range(0, n_elem):

        #Profiltypen til elementet
        profiltype = elementer[i][3]

        #Finner riktig profil til elementet
        profil_nr = index_profildata(elementer[i], profildata)
        profil    = profildata[profil_nr]

        # Sjekker om profilet er IPE
        if str(profiltype)[:1] == '1':

            # Regner h yden opp til arealsenteret
            # halve h yden p  steget + tykkelsen p  flensen (antar lik tykkelse p  begge flensene)
            z_c = profil[3] / 2 + profil[2]

            # Andre arealmoment for elementet
            I_total = aam_I_profil(profil)

            #Legger til utregningene i listen I med index = element_nr
            I[i][0] = I_total
            I[i][1] = z_c

        # Sjekker om profilet er r r
        elif str(profiltype)[:1] == '2':

            # H yden til arealsenteret er lik ytre radius
            z_c = profil[1]

            I_total = aam_ror_profil(profil)

            I[i][0] = I_total
            I[i][1] = z_c

    return I



def aam_I_profil (profildata):
    '''
    Regner ut andre arealtreghetsmoment for I-profil
```

```
    :param profildata: Profildata for et I-profil: [profiltype, bredde flens,
    tykkelse flens, h yde steg, tykkelse steg]
    :return: areal treghetsmomentet
    '''
    #H yde og tykkelse p  steget
    h_steg  = profildata[3]
    t_steg  = profildata[4]
    #Bredde og tykkelse p  flensene
    b_flens = profildata[1]
    t_flens = profildata[2]

    #Avstand fra aralsenteret til flensens arealsenter
    arm     = h_steg/2  + t_flens/2

    #Lokale areal treghetsmoment
    I_steg  = t_steg    * h_steg**3 /12
    I_flens = b_flens   * t_flens**3/12

    #Totalt areal treghetsmoment
    I_total = I_steg + 2*(I_flens + b_flens*t_flens * arm**2)

    return I_total

def aam_ror_profil(profildata):
    '''
    Regner ut andre arealtreghetsmoment for r r -profil.
    :param profildata: Profildata for et r r -profil:[profiltype, Ytre radius,
    tykkelse r r , 0, 0]
    :return: areal treghetsmoment
    '''
    #Ytre radius
    R = profildata[1]
    #Indre radius
    r = R - profildata[2]

    # Bruker formel for tykkvegget ror
    I_total = (np.pi / 4) * (R**4 - r**4)

    return I_total
```

A.8 lengder.py

```
def lengder(punkter, elementer, n_elem):
    '''
    Itererer gjennom alle elementene
    :param punkter: Liste over alle punktene med punktenes koordinater (fra
    inputfil)
    :param elementer: Liste over alle elementene med indeks p  de lokale endene (
    fra inputfil)
    :param n_elem: Antall elementer
    :return: Liste over alle elementlengdene sortert etter elementindeks
    '''
    elementlengder = np.zeros((n_elem))

    # Beregner elementlengder med Pythagoras' l resetning
    for i in range(0, n_elem):
        # OBS! Grunnet indekseringsyntaks i Python-arrays vil ikke denne funksjonen
     fungere naar vi bare har ett element.
        dx = punkter[elementer[i, 0], 0] - punkter[elementer[i, 1], 0]
        dy = punkter[elementer[i, 0], 1] - punkter[elementer[i, 1], 1]

        elementlengder[i] = np.sqrt(dx * dx + dy * dy)
    return elementlengder
```

A.9 dimensjoner_fordeltlast.py

```
ef dimensjoner_fordeltlast(profildata, laster, elementer):
    '''
```

```
   Regner rikig st rrelse  p   de fordelte lastene basert  p   at de er
    proporsjonale med bjelkenes diameter (Gitt i oppgaven)
   :param profildata: Liste over profildata for alle elementene sortert etter
    elementnummer
   :param laster: liste over alle lastene
   :return: liste over lastene med riktig st rrelse  p   de fordelte lastene
   '''
    # itererer gjennom hver last i lastlisten
    for last in laster:
        # Sjekker om lasten er en fordelt last
        if last[0] == 2:
            # Finner index til elementet lasten virker  p
            elem_nr     = int(last[1])
            elem        = elementer[elem_nr]

            # Finner index til profilet som tilh rer elementet
            profil_nr   = index_profildata(elem,profildata)
            profil      = profildata[profil_nr]

            # Kraftst rrelsene er dimensjonert for diameter  p   1m
            # F_1 er kraften i lokal ende 1, F_2 i lokal ende 2
            F_1 = last[2]
            F_2 = last[3]

            # I profildata er dimensjonene lagret som ytre diameter og  m   dobles
            D = profil[1] * 2

            last[2] = F_1 * D
            last[3] = F_2 * D

    return laster
```

A.10 index_profildata.py

```
def index_profildata(element, profildata_liste):
    '''
    Tar inn et elemento og en liste med profiler, finner profilet som h rer til
    elementet og returnerer indeksen.
    :param element: Valg element
    :param profildata_liste: Liste over profilene
    :return: Indeksen for elementets profil i profildata_liste
    '''
    type_profil = element[3]
    for i in range(len(profildata_liste)):
        if type_profil == profildata_liste[i][0]:
            return i
```

A.11 fastinnspenningskrefter.py

```
def fastinnspenningskrefter (elem_list,  laster):
    '''
    Itererer gjennom lastene, regner ut fastinnspenningskrefter og lagrer i riktig
    Element.
    :param elem_list: Liste over alle Element-objektene
    :param laster: Liste over alle lastene
    '''
    #Itererer gjennom hver last i listen "laster"
    for last in laster:
        #Lagrer indeksen til elementet lasten virker  p   og finner riktig Element-
    objekt
        elem_nr = int(last[1])
        elem    = elem_list[elem_nr]

        #Elementlengden
        l = elem.L

        #Rotasjon, antar vinkelrett pa elementet ved rotasjon = 0
        rot = np.cos(np.deg2rad( last[4]))
```

```

    #Lokal kraft i lokal ende 1 og 2
    F_1 = last[2] * rot
    F_2 = last[3] * rot


    type_last = last[0]

    #Sjekker om det er en punktlast
    if type_last ==1:
        #avstandskonstant
        a = last[5] * l
        b = l - a

        #Regner fastinnspenningsmoment i lokal ende 1 og 2 (gir 0 dersom punktlasten virker i et knutepunkt)
        m_1 =  (F_1 * a * b**2)/l**2
        m_2 = - (F_1 * b * a**2)/l**2

        #Regner ut fastinnspenningsskjerkraft i lokal ende 2 og 1
        q_2 = (m_2/l) + (m_1/l) - (F_1*a)/l
        q_1 = -F_1 - q_2

        #Fastinnspenningskrefter
        fik = np.zeros((6, 1))
        fik[1] += q_1
        fik[2] += m_1
        fik[4] += q_2
        fik[5] += m_2

        #Legger til fastinnspenningskreftene til elementets fik, men ikke dersom punktlasten virker direkte i et knutepunkt.
        if a!=0 and b!=0:
            elem.fik = elem.fik + fik

        #Legger til den negative av fastinnspenningskreftenen inn i elementets lastvektor
        elem.lokal_lastvektor = elem.lokal_lastvektor - fik

    #Fordelte laster:
    elif type_last == 2:

        #Lagrer den fordelte lasten til elementet
        elem.laster = last

        #Regner fastinnspenningsmomentene basert p  at det virker to uavhengige line re trekantlaster
        m1_F1 =  1 / 20 * F_1 * l ** 2
        m2_F1 = -1 / 30 * F_1 * l ** 2
        m1_F2 =  1 / 30 * F_2 * l ** 2
        m2_F2 = -1 / 20 * F_2 * l ** 2

        #Legger sammer resultatene for "begge" lastene F_1 og F_2
        m_1 = m1_F1 + m1_F2
        m_2 = m2_F1 + m2_F2

        #Fastinnspenningsskjerkraft
        q_1 = -(7 / 20 * F_1 + 3 / 20 * F_2) * l
        q_2 = -(3 / 20 * F_1 + 7 / 20 * F_2) * l

        #Fastinnspennings-vektor
        fik = np.zeros((6,1))
        fik[1] += q_1
        fik[2] += m_1
        fik[4] += q_2
        fik[5] += m_2

        #Lagrer til elementet
        elem.fik = elem.fik + fik

        #Lagrer lastvektoren som den negative av fastinnspenningsvektoren
        elem.lokal_lastvektor = elem.lokal_lastvektor - elem.fik
```

A.12 lastvektor.py

```python
def lastvektor_funk(n_punkter, element_liste):
    '''
    Lager den globale lastvektoren
    :param n_punkter: Antall punkter
    :param element_liste: Liste over alle Element-objektene
    :return: Global lastvektor
    '''
    lastvektor = np.zeros((3*n_punkter,1))
    n_elem = len(element_liste)


    for i in range(n_elem):
        elem = element_liste[i]

        #Lokale ender
        ende_1 = elem.ende_1
        ende_2 = elem.ende_2

        #Transformasjonsmatrise
        T = elem.T
        #Lokal lastvektor
        lvek =elem.lokal_lastvektor

        #Lokal lastvektor trasformenrt til globalt koordinatsystem
        global_lvek = np.dot(T, lvek)

        #Legger til Lokal lastvektor til
        lastvektor[ende_1*3:3*ende_1+3] += global_lvek[:3]
        lastvektor[ende_2*3:3*ende_2+3] += global_lvek[3:]



    return lastvektor
```

A.13 lokal_k.py

```python
def lokal_k (element, A,I , lengde ):
    '''

    :param element: Elementliste med elastisitetsmodul i kolonne 3 (index 2)
    :param profil: profildata til valgt element
    :param I: andre arealmoment for valgt element
    :param lengde: elementlengden
    :return: lokal stivhetsmatrise, k, 6x6
    '''

    #Elastisitetsmodul og elementlengde
    E = element[2]
    L = lengde

    #Regner ut konstanter f r det legges inn i matrisen
    EA_L = E*A/L
    EI = E*I

    #Oppretter 6x6 matrise
    lokal_k = np.zeros((6, 6))

    for i in range(6):
        #a: fortegnskonstant
        a = 1

        #Setter fortegnskonstanten lik -1 dersom i er st rre enn 2
        if i>2: a = -1

        #Legger til verdiene i lokal_k-matrisen. Hver linje legger til 2 steder i matrisen
        if i == 0 or i == 3:
            lokal_k[0][i] = a * EA_L
```

```
            lokal_k[3][i] = -a * EA_L
        elif i == 1 or i == 4:
            lokal_k[1][i] = a * 12 * EI / L ** 3
            lokal_k[2][i] = -a * 6 * EI / L ** 2
            lokal_k[4][i] = -a * 12 * EI / L ** 3
            lokal_k[5][i] = -a * 6 * EI / L ** 2
        else:
            lokal_k[1][i] = -6 * EI / L ** 2
            lokal_k[4][i] = 6 * EI / L ** 2
            lokal_k[i][i] = 4 * EI / L
            lokal_k[i][i + a * 3] = 2 * EI / L

    return lokal_k
```

A.14 system_stivhetsmatrise.py

```
def system_stivhetsmatrise( element_liste, n_punkt):
    '''
    Lager den globale system stivhetsmatrisen
    :param element_liste: Liste over Element-objektene
    :param n_punkt: Antall punkter
    :return: System stivhetsmatrise
    '''
    #Lager system stivhetsmatrisen for antallet frihetsgrader
    K = np.zeros((n_punkt *3, n_punkt *3))
    n_elem = len(element_liste)

    for i in range(n_elem):
        #Element-objektet
        elem  = element_liste[i]
        #Lokal stivhetsmatrise rotert til globalt koordinatsystem
        k_glob = elem.k_global

        #Legger til k_glob i system stivhetsmatrisne, K
        K = legg_til_K(elem,k_glob, K)

    return K



def legg_til_K (element, global_k , K):
    '''
    Legger til lokal stivhetsmatrise (som er rotert til globalt koordinatsystem)
    :param element: Element-objektet
    :param global_k: Lokal stivhetsmatrise (som er rotert til globalt koordinatsystem)
    :param K: System stivhetsmatrise
    :return: System stivhetsmatrisen K, med global_k lagt inn
    '''
    #lokal ende :
    lokal_ende_1 = element.ende_1
    lokal_ende_2 = element.ende_2

    #global ende
    global_ende_1 = 3* lokal_ende_1
    global_ende_2 = 3* lokal_ende_2


    for i in range(6):
        for j in range(6):
            # Finner riktig index for system stivhetsmatrisen
            if i >= 0 and i < 3 : rad = global_ende_1 +i
            else:                 rad = global_ende_2 +i-3

            if j >= 0 and j < 3 : col = global_ende_1 +j
            else:                 col = global_ende_2 +j-3


            input        = global_k[i][j]
            K[rad][col] += input

```

```
    return K
```

A.15 transformasjon.py

```
def transformasjon(element, punkter, lengde):
    '''
    Lager transformasjonsmatrisen for et gitt element ved  regne vinkel mellom
    lokalt koordinatsystem og det globale.
    :param element: elementet (som fra inputfilen)
    :param punkter: Liste over alle punkter
    :param lengde: Lengde p  elementet
    :return: Transformasjonsmatrisen
    '''
    #Punktdataen til elementets punkter
    punkt_1 = punkter[element[0]]
    punkt_2 = punkter[element[1]]
    #Elementlengden
    l = lengde

    #Koordinat til lokal ende (punkt)
    x_1 = punkt_1[0]
    x_2 = punkt_2[0]
    y_1 = punkt_1[1]
    y_2 = punkt_2[1]

    #Avstand mellom punktenes y-koordinater - dy og x-koordinater - dx
    dy = y_2 - y_1
    dx = x_2 - x_1
    # Vinkel i radianer

    #Transformasjonsmatrise
    t = np.zeros((2, 2))
    T = np.zeros((6,6))

    #cos (vinkel)= dx/l
    #sin(vinkel) = dy/l
    t[0][0] =   dx / l
    t[0][1] =   -dy / l
    t[1][0] =   dy / l
    t[1][1] =   dx / l

    #Setter opp den lokale transformasjonsmatrisen
    T[2][2] = T[5][5] = 1
    for i in range(2):
        for j in range(2):
            T[i][j] = T[i + 3][j + 3] = t[i][j]


    return T
```

A.16 deformasjoner.py

```
def lokal_deform (elem_liste, deformasjoner):
    '''
    Tar inn liste med alle Element-objektene, og deformasjons-vektoren. Itererer
    gjennom alle elementene og finner
    deformasjonene som h rer til elemetets pukter. Transformerer deformasjonene
    til lokalt koordinatsystem og lagrer
    til elementet
    :param elem_liste: Liste med alle Element-objektene
    :param deformasjoner: Liste over alle deformasjoner (og rotasjoner) for
    knutepunktene i forhold til globalt koordinatsystem
    '''
    for i in range(len(elem_liste)):
        #Element-objektet
        elem = elem_liste[i]
        #Transformasjonsmatrise
        T = elem.T
```

```

        #Lokale ender
        ende1 = elem.ende_1
        ende2 = elem.ende_2

        #Indeks til de lokale endene i lastvektoren
        pkt1 = ende1*3
        pkt2 = ende2*3

        #Henter ut deformasjonene for elementets punkter
        def_1 = deformasjoner[pkt1:pkt1+3]
        def_2 = deformasjoner[pkt2:pkt2+3]

        #Sl r sammen deformasjoner for begge punkter til en vektor/liste
        lok_def = np.zeros((6,1))
        lok_def[0:3] = def_1
        lok_def[3:] = def_2

        #Lagrer til elemetet
        elem.deformasjoner = np.linalg.solve(T, lok_def)


def deformasjoner (K, lastvektor):
    '''
    Regner deformasjonene for systemet
    :param K: System stivhetsmatrise
    :param lastvektor: System/global lastvektor
    :return: Deformasjoner (og rotasjoner)
    '''
    d = np.linalg.solve(K, lastvektor)
    return d
```

A.17 randbetingelser.py

```
def randbetingelser (n_punkter, punkter, K, last_vektor):
    '''
    Innf rer en h y fjerstivhet i diagonalen p   system stivhetsmatrisen i alle
    frihetsgradene til punktene som er
    fast innspent
    :param n_punkter: Antall punkter
    :param punkter: Liste over alle punktene
    :param K: System stivhetsmatrisen
    :param last_vektor: Global lastvektor
    :return: System stivhetsmatrisen, K, og lastvektoren med innf rt
    randbetingeler
    '''
    #ittererer gjennom alle frihetsgradene
    for i in range(n_punkter*3):
        # Heltallsdivisjon   finne element nummeret
        elem_nr = i//3

        #sjekker om punktet er fast innspent (=1)
        if punkter[elem_nr][2] == 1:

            #Legger til fjerstivhet
            K[i][i] = K[i][i]+10e8

            #Setter lastvektoren til frihetsgraden lik null
            last_vektor[i] = 0


    return K, last_vektor
```

A.18 el_krefter.py

```
def el_krefter (elem_liste ):
    '''
    Regner ut element-kreftene
```

```
    :param elem_liste: liste over Element-objektene
    '''

    for i in range(len(elem_liste)):
        #Element-objektet
        elem = elem_liste[i]

        #Lokale deformasjoner
        v = elem.deformasjoner
        #Lokal stivhetsmatrise for lokalt koordinatsystem
        k = elem.k_lokal
        # Lokal fastinnspenningskrefter
        fik = elem.fik

        #Regner ut kreftene
        S = k.dot(v) + fik

        #Lagrer kreftene til elementet
        elem.krefter = S
```

A.19 midtmoment.py

```
 def midt_moment(elem_list, ):


    for i in range(len(elem_list)):
        elem = elem_list[i]
        krefter = elem.krefter
        last = elem.laster
        l = elem.L

        m_1 =  krefter[2][0]
        m_2 =  krefter[5][0]
        q_1 =  krefter[1][0]
        q_2 =  krefter[4][0]

        m_m = 0
        q_m = 0



        if len(last)==0:
            m_2 = m_2*-1
            m_m +=( m_1 + m_2 )/2
            q_m += (q_2 - q_1) /2

        else:
            if last[0] == 2:
                F_2 = last[3]
                F_1 = last[2]
                dF = (F_2 - F_1) / 2
                if F_2==F_1:
                    m_m += F_1 * l ** 2 / 8 - q_1 * l / 2
                    q_m  = q_1 - F_1 * l/2
                elif F_1>F_2:
                    m_m = m_2 - q_2 * l / 2 + F_2 * l ** 2 / 8 - dF * l ** 2 / 24
                    q_m = q_1 - F_1 * l / 2 - dF * l / 4
                else:
                    m_m = m_1 + q_1 * l / 2 - F_1 * l ** 2 / 8 - dF * l ** 2 / 24
                    q_m = q_1 - F_1 * l / 2 - dF * l / 4

        elem.M_midt = m_m
        elem.Q_midt = q_m



def mom(elem_list, laster):
    for l in laster:
        if l[0]==2:
            elem_nr = int(l[1])
            e = elem_list[elem_nr]
```

```
            F_1 = l[2]
            F_2 = l[3]
            l = e.L
            endemom = e.krefter[2::3]
            skjer = e.krefter[1::3]

            if F_1 ==F_2:
                m_m = (-F_1 * l**2)/8
                e.M_midt -= m_m
            else:
                m_1 = F_1 *l**2/48
                m_2 = F_2 *l**2 *5/48
                m_3 = endemom[1][0]
                m_4 = skjer[1][0]
                m_m = m_1 + m_2 +m_3 -m_4
                e.M_midt = m_m
```

A.20 maks_boyespenning.py

```
 def maks_boyespenning(elem_list):
    '''
    OBS! Regner feil!
    Regner boyespenning i p  midten og i hver ende av hvert element. Lagrer den
    absolutte st rste spenningen til elementet.
    :param elem_list: Liste over Element-objektene
    '''

    for e in elem_list:
        #Areal
        A = e.A
        #Andre arealtreghetsmoment
        I   = e.I
        #Avstand til arealsenter i tverrsnittet
        z_c = e.z_c

        #Moment p  midten
        M_midt = e.M_midt
        #Elemenkreftene
        krefter = e.krefter
        #Normal og Momentkreftene
        N = krefter[0][0]
        M = krefter[2::3]

        #Utregning av boyespenning
        sigma_1  =N/A+ M[0][0]/I * z_c
        sigma_2  =N/A+ M[1][0]/I * z_c
        #Noe feil med utregningen av midt moment
        sigma_m  =0 #N/A+ M_midt/I * z_c

        #sjekker hvilken som er absolutt st rst
        sigma_max = max(abs(sigma_m), abs(sigma_1),abs(sigma_2))

        #Lagrer til elementet
        e.sigma_max = sigma_max/1e6


def prosent_av_flyt(flytspennning, elem_list):
    '''
    Regner b yespenningen i prosent av gitt flytspenning, og lagrer til elementet
    :param flytspennning:
    :param elem_list: liste over Element-objektene
    '''
    fy = flytspennning

    for e in elem_list:
        maks_spenning = e.sigma_max
        prosent = maks_spenning / fy *100

        #Lagrer prosenten av flyt i elementet med en desimal
        e.prosent_av_flyt = round(prosent, 1)
```

A.21 printfunksjoner.py

```
 def print_deformasjoner(rot, x_def, y_def, n_punkter):
    deformasjoner = PrettyTable()

    deformasjoner.field_names = ["Knutepunkt", "X-forskyvning", "Y-forskyvning", "
    Rotasjon"]
    for i in range(n_punkter):
        deformasjoner.add_row([i , round(x_def[i][0],5) ,round(y_def[i][0], 5),
    round(rot[i][0],5)/1e3])
        deformasjoner.align = 'r'
    print(deformasjoner)


def print_knutepunktskrefter(elem_liste):
    krefter = PrettyTable()
    krefter.field_names = ["Element", "N_1" , "Q_1", "M_1 ","N_2" , "Q_2", "M_2 " ]
    for i in range(len(elem_liste)):
        e = elem_liste[i]
        kr = e.krefter/ 1e3
        k = np.around(kr, 1)
        krefter.add_row([i,k[0][0],k[1][0],k[2][0],k[3][0],k[4][0],k[5][0] ])

        krefter.align = 'r'
    print(krefter)

def print_opplager(elem_liste, punkter):
    opplager = PrettyTable()
    opplager.field_names= ["Knutepunkt", "N [kN]", "Q [kN]", "M [kNm]"]
    for i in range(len(punkter)):
        punkt = punkter[i]

        N = 0
        Q = 0
        M = 0
        if punkt[2] ==1:
            for e in elem_liste:
                krefter = np.zeros((3,1))
                if e.ende_1 == i:
                    krefter = e.krefter[:3]
                elif e.ende_2 ==i:
                    krefter = e.krefter[3:]
                N += krefter[0][0]
                Q += krefter[1][0]
                M += krefter[2][0]

            opplager.add_row([i, round(N/1e3, 2),round( Q/1e3, 2),round( M/1e3, 2)
    ])
    print(opplager)

def print_M ( elem_list):
    midt_m = PrettyTable( )
    midt_m.field_names = ["Element", "M 1 [kNm]",  "M midt [kNm]", "M 2[kNm]"]
    for i in range(len(elem_list)):
        e = elem_list[i]
        moment = e.krefter[2::3]/1e3
        M_midt = e.M_midt/1e3

        midt_m.add_row([i, round(moment[0][0], 1), round(M_midt,1), round( moment
    [1][0], 1)])
    print(midt_m)

def print_Q ( elem_list):
    midt_q = PrettyTable( )
    midt_q.field_names = ["Element", "Q 1 [kN]", "Q midt [kN]", "Q 2[kN]"]
    for i in range(len(elem_list)):
        e = elem_list[i]
        skjer = e.krefter[1::3]/1e3
        Q_midt = e.Q_midt /1e3

        midt_q.add_row([i, round(skjer[0][0], 1), round(Q_midt, 1),round( skjer
    [1][0], 1)])
    print(midt_q)
```

```

def print_maks_boyespenning(elem_list, flytspenning):
    sigma = PrettyTable()
    sigma.field_names = ["Element nummer", "Maks boyespenning", "Prosent av flyt  = " + str(flytspenning) +"MPa"]
    for i in range(len(elem_list)):
        e = elem_list[i]
        sigma.add_row([i, round(e.sigma_max, 1), str(e.prosent_av_flyt) + "%"])
    print(sigma)
```

A.22 structure_visualization.py

```
 def setup_plots():
    fig_init, ax_init = plt.subplots()
    fig_def, ax_def = plt.subplots()
    ax_init.set_title('Initialramme')
    ax_def.set_title('Deformert ramme')
    ax_init.axes.set_aspect('equal')
    ax_def.axes.set_aspect('equal')
    return fig_init, ax_init, fig_def, ax_def


def plot_structure(ax, punkt, elem, numbers, index_start):
    # This is a translation of the original function written by Josef Kiendl in Matlab
    # It has been slightly modified in order to be used in TMR4176

    # This function plots the beam structure defined by nodes and elements
    # The bool (0 or 1) 'numbers' decides if node and element numbers are plotted or not

    # Change input to the correct format
    nodes = np.array(punkt[:, 0:2], copy=1, dtype=int)
    el_nod = np.array(elem[:, 0:2], copy=1, dtype=int) + 1

    # Start plotting part
    for iel in range(0, el_nod.shape[0]):
        # Plot element
        ax.plot([nodes[el_nod[iel, 0] - 1, 0], nodes[el_nod[iel, 1] - 1, 0]],
                [nodes[el_nod[iel, 0] - 1, 1], nodes[el_nod[iel, 1] - 1, 1]], '-k', linewidth=2)

        if numbers == 1:
            # Plot element numbers. These are not plotted in the midpoint to
            # avoid number superposition when elements cross in the middle
            ax.text(nodes[el_nod[iel, 0] - 1, 0] + (nodes[el_nod[iel, 1] - 1, 0] - nodes[el_nod[iel, 0] - 1, 0]) / 2.5,
                    nodes[el_nod[iel, 0] - 1, 1] + (nodes[el_nod[iel, 1] - 1, 1] - nodes[el_nod[iel, 0] - 1, 1]) / 2.5,
                    str(iel + index_start), color='blue', fontsize=16)

    if numbers == 1:
        # Plot node number
        for inod in range(0, nodes.shape[0]):
            ax.text(nodes[inod, 0], nodes[inod, 1], str(inod + index_start), color='red', fontsize=16)


def plot_structure_def(ax, punkt, elem, numbers, index_start, r):
    # This is a translation of the original function written by Josef Kiendl in Matlab
    # This function plots the deformed beam structure defined by nodes and elements
    # The bool (0 or 1) 'numbers' decides if node and element numbers are plotted or not

    # Change input to the correct format
    nodes = np.array(punkt[:, 0:2], copy=1, dtype=int)
    el_nod = np.array(elem[:, 0:2], copy=1, dtype=int) + 1
    nod_dof = np.arange(1, nodes.shape[0] + 1, 1, dtype=int)

    if numbers == 1:
```

```
        # Plot node number
        for inod in range(0, nodes.shape[0]):
            ax.text(nodes[inod, 0], nodes[inod, 1], str(inod + index_start), color='red', fontsize=16)

    for iel in range(0, el_nod.shape[0]):
        delta_x = nodes[el_nod[iel, 1] - 1, 0] - nodes[el_nod[iel, 0] - 1, 0]
        delta_z = nodes[el_nod[iel, 1] - 1, 1] - nodes[el_nod[iel, 0] - 1, 1]
        L = np.sqrt(delta_x ** 2 + delta_z ** 2)
        if delta_z >= 0:
            psi = np.arccos(delta_x / L)
        else:
            psi = -np.arccos(delta_x / L)

        phi = np.zeros((2, 1))
        for inod in range(0, 2):
            if nod_dof[el_nod[iel, inod] - 1] > 0:
                phi[inod] = r[nod_dof[el_nod[iel, inod] - 1] - 1]
        x = np.array([0, L])
        z = np.array([0, 0])
        xx = np.arange(0, 1.01, 0.01) * L
        cs = CubicSpline(x, z, bc_type=((1, -phi[0, 0]), (1, -phi[1, 0])))
        zz = cs(xx)

        # Rotate
        xxzz = np.array([[np.cos(psi), -np.sin(psi)], [np.sin(psi), np.cos(psi)]]) @ np.vstack([xx, zz])

        # Displace
        xx2 = xxzz[0, :] + nodes[el_nod[iel, 0] - 1, 0]
        zz2 = xxzz[1, :] + nodes[el_nod[iel, 0] - 1, 1]
        ax.plot(xx2, zz2, '-k', linewidth=2)

        if numbers == 1:
            # Plot element numbers. These are not plotted in the midpoint to
            # avoid number superposition when elements cross in the middle
            ax.text(xx2[round(xx2.size / 2.5)], zz2[round(xx2.size / 2.5)], str(iel + index_start), color='blue',
                    fontsize=16)
```

B Input-filer

B.1 input.txt

```
13
0 0 1
16.5 0 0
33 0 1
2.5 25 0
16.5 25 0
30.5 25 0
5 50 0
16.5 50 0
28 50 0
7.5 75 0
25.5 75 0
7.5 93 0
25.5 93 0
22
0 1 210000000 21
1 2 210000000 21
0 3 210000000 23
1 3 210000000 22
1 5 210000000 22
2 5 210000000 23
3 4 210000000 21
4 5 210000000 21
3 6 210000000 23
4 6 210000000 22
4 8 210000000 22
5 8 210000000 23
6 7 210000000 21
7 8 210000000 21
6 9 210000000 23
7 9 210000000 22
7 10 210000000 22
8 10 210000000 23
9 10 210000000 21
9 11 70000000 11
10 12 70000000 11
11 12 70000000 12
9
1 21 4000000 0 0 0
1 21 1000000 0 0 0.5
1 21 4000000 0 0 1
2 14 540000 180000 0 0
2 15 360000 120000 0 0
2 16 360000 120000 0 0
2 17 540000 180000 0 0
2 8 180000 0 0 0
2 11 180000 0 0 0
5
11 0.45 0.027 0.90 0.02
12 0.3 0.04 0.65 0.024
21 0.4 0.25 0 0
22 0.5 0.25 0 0
23 0.75 0.15 0 0
300
```

B.2 inputforklaring.txt

```
#Antall knutepunkt
13
#x-koordinat, y-koordinat, randbetingelse (0 = fri rotasjon, 1 = null rotasjon)
0 0 1
16.5 0 0
33 0 1
2.5 25 0
16.5 25 0
30.5 25 0
```

```
5 50 0
16.5 50 0
28 50 0
7.5 75 0
25.5 75 0
7.5 93 0
25.5 93 0
#Antall elementer
22
#Knutepunkt start, knutepunkt ende, E-modul [MPa], profiltype (1 = IPE, 2 = ror)
0 1 210000000 21
1 2 210000000 21
0 3 210000000 23
1 3 210000000 22
1 5 210000000 22
2 5 210000000 23
3 4 210000000 21
4 5 210000000 21
3 6 210000000 23
4 6 210000000 22
4 8 210000000 22
5 8 210000000 23
6 7 210000000 21
7 8 210000000 21
6 9 210000000 23
7 9 210000000 22
7 10 210000000 22
8 10 210000000 23
9 10 210000000 21
9 11 70000000 11
10 12 70000000 11
11 12 70000000 12
#Antall laster
9
#Type last (1 = punkt, 2 = fordelt), st rrelse last 1, st rrelse last 2 (0 for
    punktlast), rotasjon, avstandskonstant
1 21 4000000 0 0 0
1 21 1000000 0 0 0.5
1 21 4000000 0 0 1
2 14 540000 180000 0 0
2 15 360000 120000 0 0
2 16 360000 120000 0 0
2 17 540000 180000 0 0
2 8 180000 0 0 0
2 11 180000 0 0 0
#Antall tverrsnitt
5
#Type tverrsnitt (1 = IPE, 2 = ror), bredde flens/ytre radius, tykkelse flens/
    tykkelse ror, h yde steg/0 for ror, tykkelse steg/0 for ror
11 0.45 0.027 0.90 0.02
12 0.3 0.04 0.65 0.024
21 0.4 0.25 0 0
22 0.5 0.25 0 0
23 0.75 0.15 0 0
#Flytspenning [MPa]
300
```

C Resultatfil

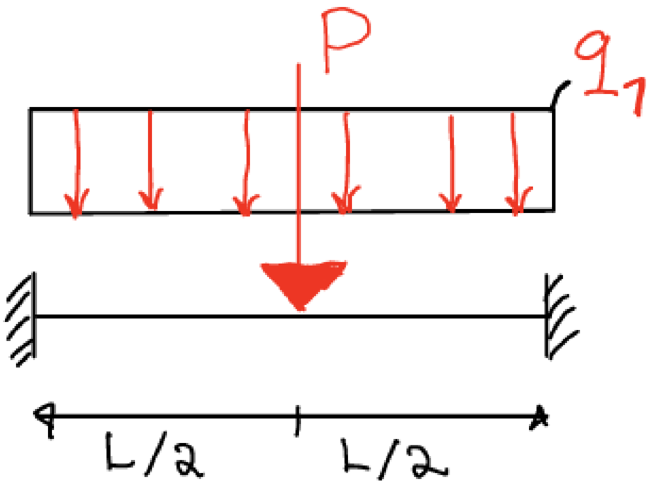
```
+------------+--------------+--------------+------------------------+
| Knutepunkt | X-forskyvning | Y-forskyvning |               Rotasjon |
+------------+--------------+--------------+------------------------+
|          0 |      -0.0168 |     -0.05448 |              -3.89e-06 |
|          1 |     -2.05336 |      0.85298 | -0.0003467700000000004 |
|          2 |      -0.0177 |      0.06348 |              -3.71e-06 |
|          3 |    -28.02097 |     -7.60121 |            -0.00160518 |
|          4 |    -29.92251 |      1.70884 |            -0.00032276 |
|          5 |    -28.02617 |       9.3157 |  -0.0015967400000000001 |
|          6 |    -70.84368 |     -9.86756 |            -0.00361255 |
|          7 |    -71.80315 |      2.56576 |            -0.00719717 |
|          8 |    -70.83358 |     13.28788 |            -0.00352564 |
|          9 |    -107.3558 |     -7.48661 |             0.00380945 |
|         10 |   -107.30608 |     12.61173 |   0.0035308799999999997 |
|         11 |    -74.46821 |     19.93049 |            -0.01781682 |
|         12 |    -73.51126 |     39.90588 |             0.01788565 |
+------------+--------------+--------------+------------------------+
+---------+----------+---------+----------+----------+---------+----------+
| Element |      N_1 |     Q_1 |     M_1  |      N_2 |     Q_2 |     M_2  |
+---------+----------+---------+----------+----------+---------+----------+
|       0 |  11196.5 |    22.0 |    -95.1 | -11196.5 |   -22.0 |   -267.1 |
|       1 | -11191.6 |    40.7 |   -421.8 |  11191.6 |   -40.7 |   -249.7 |
|       2 |  54746.1 |  -160.9 |   3984.5 | -54746.1 |   160.9 |     57.2 |
|       3 | -22930.7 |     5.8 |    341.3 |  22930.7 |    -5.8 |   -507.6 |
|       4 |  22909.6 |     5.2 |    347.6 | -22909.6 |    -5.2 |   -495.6 |
|       5 | -63772.0 |  -159.5 |   3957.7 |  63772.0 |   159.5 |     50.6 |
|       6 |  12321.1 |    75.8 |   -909.4 | -12321.1 |   -75.8 |   -151.2 |
|       7 | -12287.4 |   105.5 |   -362.0 |  12287.4 |  -105.5 |  -1115.2 |
|       8 |  34648.4 | -1042.9 |   1359.8 | -34648.4 | -1218.3 | -13032.3 |
|       9 | -29597.1 |    67.5 |    226.4 |  29597.1 |   -67.5 |  -2084.3 |
|      10 |  29567.4 |    60.9 |    286.8 | -29567.4 |   -60.9 |  -1962.8 |
|      11 | -43665.7 | -1066.5 |   1560.2 |  43665.7 | -1194.7 | -12639.4 |
|      12 |   7568.5 |  1623.7 |  -8046.4 |  -7568.5 | -1623.7 | -10626.6 |
|      13 |  -7648.1 |  1663.3 | -10885.3 |   7648.1 | -1663.3 |  -8242.6 |
|      14 |   6720.9 | -6337.1 |  23163.0 |  -6720.9 | -2707.8 |   3491.3 |
|      15 | -12030.9 | -3775.8 |  10859.3 |  12030.9 | -2601.2 |  -9374.2 |
|      16 |  11980.5 | -3752.6 |  10652.6 | -11980.5 | -2624.4 |  -9783.5 |
|      17 | -15719.9 | -6276.1 |  22844.9 |  15719.9 | -2768.8 |   2276.8 |
|      18 |   -250.6 |  -733.8 |   6667.8 |    250.6 |   733.8 |   6539.7 |
|      19 |  -4510.1 |   147.4 |   -785.0 |   4510.1 |  -147.4 |  -1867.7 |
|      20 |  -4489.9 |  -147.4 |    967.0 |   4489.9 |   147.4 |   1685.7 |
|      21 |   -147.4 |  -510.1 |   1867.7 |    147.4 |  -489.9 |  -1685.7 |
+---------+----------+---------+----------+----------+---------+----------+
+------------+----------+---------+---------+
| Knutepunkt |  N [kN]  |  Q [kN] | M [kNm] |
+------------+----------+---------+---------+
|     0      | 65942.68 | -138.92 | 3889.45 |
|     2      | -52580.4 | -200.23 | 3707.99 |
+------------+----------+---------+---------+
+---------+----------+-------------+----------+
| Element | M 1 [kNm] | M midt [kNm] | M 2[kNm] |
+---------+----------+-------------+----------+
|    0    |   -95.1   |     86.0     |  -267.1  |
|    1    |   -421.8  |    -86.1     |  -249.7  |
|    2    |   3984.5  |    1963.7    |   57.2   |
|    3    |   341.3   |    424.4     |  -507.6  |
|    4    |   347.6   |    421.6     |  -495.6  |
|    5    |   3957.7  |    1953.5    |   50.6   |
|    6    |   -909.4  |    -379.1    |  -151.2  |
|    7    |   -362.0  |    376.6     | -1115.2  |
|    8    |   1359.8  |   -9446.8    | -13032.3 |
|    9    |   226.4   |    1155.3    | -2084.3  |
|   10    |   286.8   |    1124.8    | -1962.8  |
|   11    |   1560.2  |   -9077.5    | -12639.4 |
|   12    |  -8046.4  |    1290.1    | -10626.6 |
|   13    |  -10885.3 |   -1321.4    | -8242.6  |
|   14    |  23163.0  |   25136.7    |  3491.3  |
|   15    |  10859.3  |    7347.0    | -9374.2  |
|   16    |  10652.6  |    6960.8    | -9783.5  |
|   17    |  22844.9  |   23983.1    |  2276.8  |
|   18    |   6667.8  |     64.1     |  6539.7  |
```

```
|    19    |   -785.0  |     541.4     | -1867.7  |
|    20    |   967.0   |     -359.3    |  1685.7  |
|    21    |   1867.7  |     1776.7    | -1685.7  |
+----------+-----------+---------------+----------+
+----------+----------+-------------+---------+
| Element | Q 1 [kN] | Q midt [kN] | Q 2[kN] |
+----------+----------+-------------+---------+
|    0     |   22.0   |    -22.0    |  -22.0  |
|    1     |   40.7   |    -40.7    |  -40.7  |
|    2     |  -160.9  |    160.9    |  160.9  |
|    3     |   5.8    |     -5.8    |   -5.8  |
|    4     |   5.2    |     -5.2    |   -5.2  |
|    5     |  -159.5  |    159.5    |  159.5  |
|    6     |   75.8   |    -75.8    |  -75.8  |
|    7     |  105.5   |    -105.5   |  -105.5 |
|    8     | -1042.9  |    -87.7    | -1218.3 |
|    9     |   67.5   |    -67.5    |  -67.5  |
|    10    |   60.9   |    -60.9    |  -60.9  |
|    11    | -1066.5  |    -64.1    | -1194.7 |
|    12    |  1623.7  |   -1623.7   | -1623.7 |
|    13    |  1663.3  |   -1663.3   | -1663.3 |
|    14    | -6337.1  |    1814.6   | -2707.8 |
|    15    | -3775.8  |    587.3    | -2601.2 |
|    16    | -3752.6  |    564.1    | -2624.4 |
|    17    | -6276.1  |    1753.6   | -2768.8 |
|    18    |  -733.8  |    733.8    |  733.8  |
|    19    |  147.4   |    -147.4   |  -147.4 |
|    20    |  -147.4  |    147.4    |  147.4  |
|    21    |  -510.1  |     10.1    |  -489.9 |
+----------+----------+-------------+---------+
+----------------+-------------------+---------------------------+
| Element nummer | Maks boyespenning | Prosent av flyt  = 300MPa |
+----------------+-------------------+---------------------------+
|       0        |        24.0       |            8.0%           |
|       1        |        34.5       |           11.5%           |
|       2        |       106.4       |           35.5%           |
|       3        |        44.4       |           14.8%           |
|       4        |        42.7       |           14.2%           |
|       5        |       100.0       |           33.3%           |
|       6        |        25.5       |            8.5%           |
|       7        |        51.1       |           17.0%           |
|       8        |        61.4       |           20.5%           |
|       9        |        72.9       |           24.3%           |
|       10       |        53.3       |           17.8%           |
|       11       |       133.2       |           44.4%           |
|       12       |       198.2       |           66.1%           |
|       13       |       238.6       |           79.5%           |
|       14       |       129.0       |           43.0%           |
|       15       |       122.3       |           40.8%           |
|       16       |       136.1       |           45.4%           |
|       17       |        92.1       |           30.7%           |
|       18       |       134.7       |           44.9%           |
|       19       |       245.0       |           81.7%           |
|       20       |        34.5       |           11.5%           |
|       21       |       196.2       |           65.4%           |
+----------------+-------------------+---------------------------+
```

D Hndberegninger

D.1 Testkonstruksjon 1 - fast innspent bjelke

Testkonstruksjon - 1



Løser for følgende:

$L = 4m$

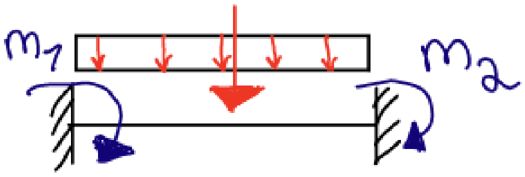
$P = 100kN$

$q = 10kN/m$

Fast innspent bjelke 1 av 4

Tar i bruk kjente løsninger for henholdvis en bjelke med en punktlast på midten og en jevnt fordelt last.

Fastinnspenningsmoment:



$$\begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = \underbrace{\frac{Pl}{8}\begin{bmatrix} -1 \\ 1 \end{bmatrix}}_{\text{fra punktlast}} + \underbrace{\frac{ql^2}{12}\begin{bmatrix} -1 \\ 1 \end{bmatrix}}_{\text{fra jevnt fordelt last}}$$

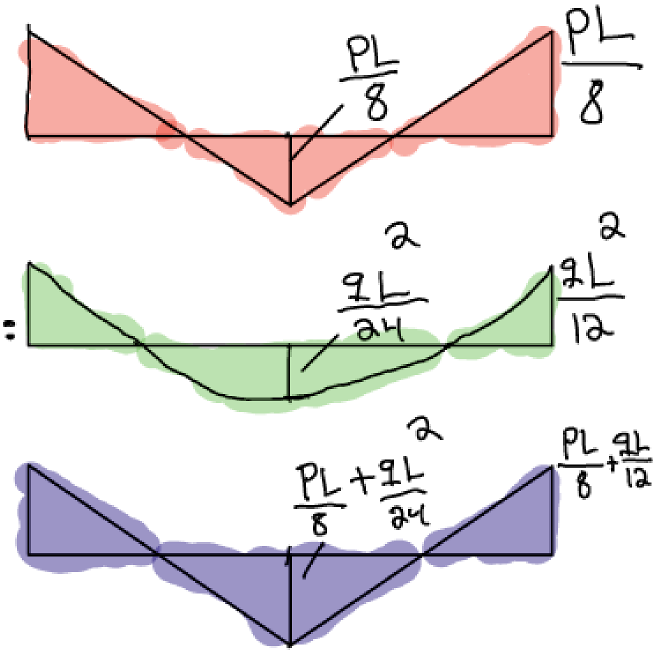
$$m_1 = -\frac{PL}{8} - \frac{ql^2}{12}, \quad m_2 = \frac{Pl}{8} + \frac{ql^2}{12}$$

Fast innspent bjelke 2 av 4

Punktlast:

Jevnt fordelt last:

endelig resultat:



$$M(0) = -\frac{100\cdot 4m}{8} - \frac{10\cdot 4^2}{24} = \underline{-63{,}3\,kNm}$$

$$M\left(\frac{L}{4}\right) = \underline{0}$$

$$M\left(\frac{L}{2}\right) = \frac{100}{8} + \frac{10\cdot 4^2}{24} = \underline{56{,}6\,kNm}$$

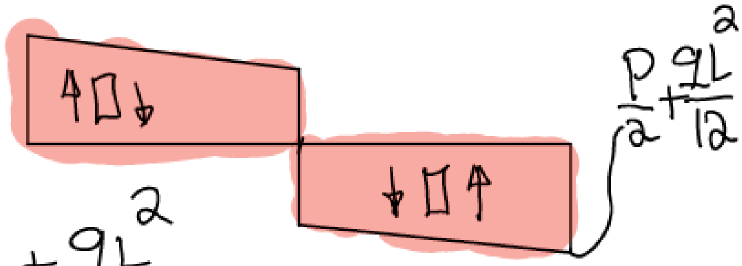
$$M\left(\frac{3L}{4}\right) = \underline{0}$$

$$M(L) = \underline{63{,}3\,kNm}$$

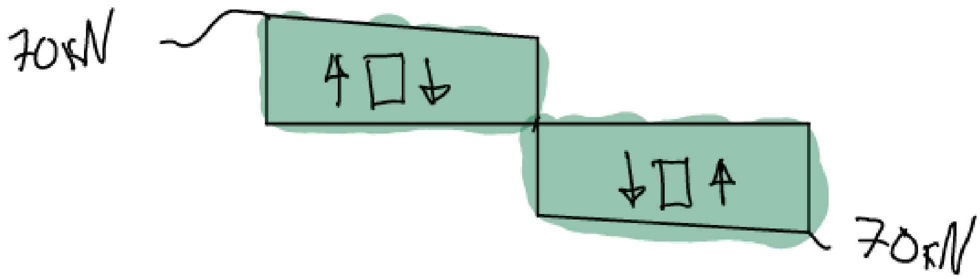
Fast innspent bjelke 3 av 4

Vi ser bort fra aksialdiagram siden vi har ingen horisontale krefter.

Vi kan regne skjærkraft utfra det vi vet om lasttilfellene

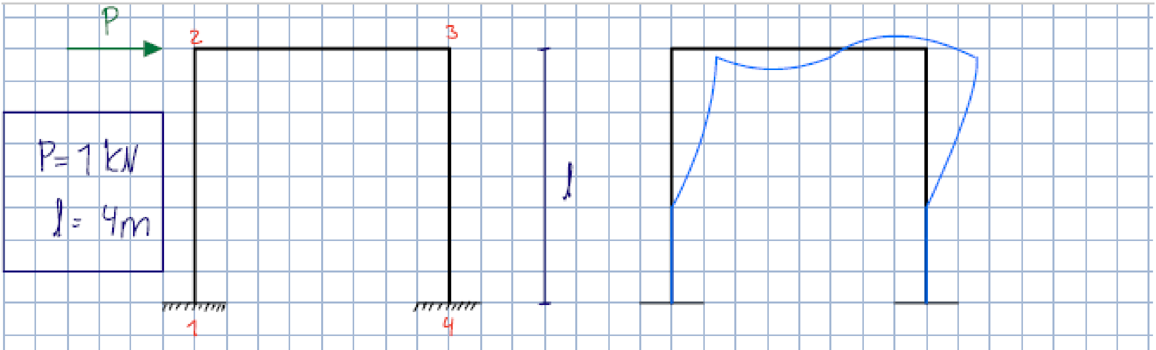


$$V(L) = \frac{P}{2} + \frac{qL^2}{12}$$



Fast innspent bjelke 4 av 4

D.2 Testkonstruksjon 2 - portalramme



Setter opp de bidragene stivhetsmatrisen til et rammeprogram generelt vil inneholde, for så å stryke det av liten eller ingen betydning.

Vet $\theta_1 = 0$, $w_1 = 0$ i tillegg til at vi ser bort fra aksialdeformasjoner som betyr $u_1 = u_2 = 0$

$$
\begin{bmatrix} N_1 \\ Q_1 \\ M_1 \\ N_2 \\ Q_2 \\ M_2 \end{bmatrix}
=
\begin{bmatrix}
\frac{EA}{l} & 0 & 0 & -\frac{EA}{l} & 0 & 0 \\
0 & \frac{12EI}{l^3} & -\frac{6EI}{l^2} & 0 & -\frac{12EI}{l^3} & -\frac{6EI}{l^2} \\
0 & -\frac{6EI}{l^2} & \frac{4EI}{l} & 0 & \frac{6EI}{l^2} & \frac{2EI}{l} \\
-\frac{EA}{l} & 0 & 0 & \frac{EA}{l} & 0 & 0 \\
0 & -\frac{12EI}{l^3} & \frac{6EI}{l^2} & 0 & \frac{12EI}{l^3} & \frac{6EI}{l^2} \\
0 & -\frac{6EI}{l^2} & \frac{2EI}{l} & 0 & \frac{6EI}{l^2} & \frac{4EI}{l}
\end{bmatrix}
\begin{bmatrix} u_1 \\ w_1 \\ \theta_1 \\ u_2 \\ w_2 \\ \theta_2 \end{bmatrix}
+
\begin{bmatrix} 0 \\ q_1 \\ m_1 \\ 0 \\ q_2 \\ m_2 \end{bmatrix}
$$

$$
\begin{bmatrix} Q_1 \\ M_2 \end{bmatrix}
=
\begin{bmatrix} \frac{12EI}{l^3} & \frac{6EI}{l^2} \\ \frac{6EI}{l^2} & \frac{4EI}{l} \end{bmatrix}
\begin{bmatrix} w_2 \\ \theta_2 \end{bmatrix}
+
\begin{bmatrix} P \\ 0 \end{bmatrix}
$$

$$
\begin{bmatrix} Q_3 \\ M_3 \end{bmatrix}
=
\begin{bmatrix} \frac{12EI}{l^3} & \frac{6EI}{l^2} \\ \frac{6EI}{l^2} & \frac{4EI}{l} \end{bmatrix}
\begin{bmatrix} w_3 \\ \theta_3 \end{bmatrix}
$$

portalramme 1 av 5

$$M_2 = \frac{4EI}{\ell}\left(\theta_2 + \frac{1}{2}\theta_3\right)$$

$$M_3 = \frac{4EI}{\ell}\left(\theta_3 + \frac{1}{2}\theta_2\right)$$

Hjørne 2:

$$\sum M_2 = 0 \Rightarrow \frac{6EI}{\ell^2}w_2 + \frac{4EI}{\ell}\theta_2 + \frac{4EI}{\ell}\theta_2 + \frac{2EI}{\ell}\theta_3$$

$$= \frac{6EI}{\ell^2}w_2 + \frac{8EI}{\ell}\theta_2 + \frac{2EI}{\ell}\theta_3 = 0$$

Hjørne 3:

$$\sum M_3 = 0 \Rightarrow \frac{6EI}{\ell^2}w_3 + \frac{4EI}{\ell}\theta_3 + \frac{4EI}{\ell}\theta_3 + \frac{2EI}{\ell}\theta_2$$

$$= \frac{6EI}{\ell^2}w_3 + \frac{8EI}{\ell}\theta_3 + \frac{2EI}{\ell}\theta_2 = 0$$

$$\sum Q = 0 \Rightarrow \frac{12EI}{\ell^3}w_2 + \frac{6EI}{\ell^2}\theta_2 + \frac{12EI}{\ell^3}w_3 + \frac{6EI}{\ell^2}\theta_3 + P = 0$$

$$\boxed{w_2 = w_3 = w}$$

$$\Rightarrow \sum Q = \frac{24EI}{\ell^3}w + \frac{6EI}{\ell^2}\theta_2 + \frac{6EI}{\ell^2}\theta_3$$

$$\begin{bmatrix} \frac{8EI}{\ell} & \frac{2EI}{\ell} & \frac{6EI}{\ell^2} \\ \frac{2EI}{\ell} & \frac{8EI}{\ell} & \frac{6EI}{\ell^2} \\ \frac{6EI}{\ell^2} & \frac{6EI}{\ell^2} & \frac{24EI}{\ell^3} \end{bmatrix} \begin{bmatrix} \theta_2 \\ \theta_3 \\ w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -P \end{bmatrix}$$

portalramme 2 av 5

$$\frac{2EI}{l}\begin{bmatrix} 4 & 1 & 3 \\ 1 & 4 & 3 \\ 3 & 3 & 12 \end{bmatrix}\begin{bmatrix} \theta_2 \\ \theta_3 \\ \frac{w}{l} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -Pl \end{bmatrix}$$

Stokker om på ligningen og får:

$$\frac{2EI}{l}\begin{bmatrix} 12 & 3 & 3 \\ 3 & 4 & 1 \\ 3 & 1 & 4 \end{bmatrix}\begin{bmatrix} \frac{w}{l} \\ \theta_3 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} -Pl \\ 0 \\ 0 \end{bmatrix}$$

$$\left[\begin{array}{ccc|c} 12 & 3 & 3 & -1 \\ 3 & 4 & 1 & 0 \\ 3 & 1 & 4 & 0 \end{array}\right] \sim \left[\begin{array}{ccc|c} 1 & 0 & 0 & -\frac{5}{42} \\ 0 & 1 & 0 & \frac{1}{14} \\ 0 & 0 & 1 & \frac{1}{14} \end{array}\right]$$

$$\frac{2EI}{l}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \frac{w}{l} \\ \theta_3 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} -\frac{5}{42} \\ \frac{1}{14} \\ \frac{1}{14} \end{bmatrix}$$

$$\begin{bmatrix} \frac{w}{l} \\ \theta_3 \\ \theta_2 \end{bmatrix} = \frac{Pl^2}{2EI}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} -\frac{5}{42} \\ \frac{1}{14} \\ \frac{1}{14} \end{bmatrix}$$

portalramme 3 av 5

$$w = -\frac{5}{84}\frac{Pl^3}{EI}$$

$$\theta_3 = \frac{Pl}{EI}\,\frac{1}{28}$$

$$\theta_2 = \frac{Pl^2}{EI}\,\frac{1}{28}$$

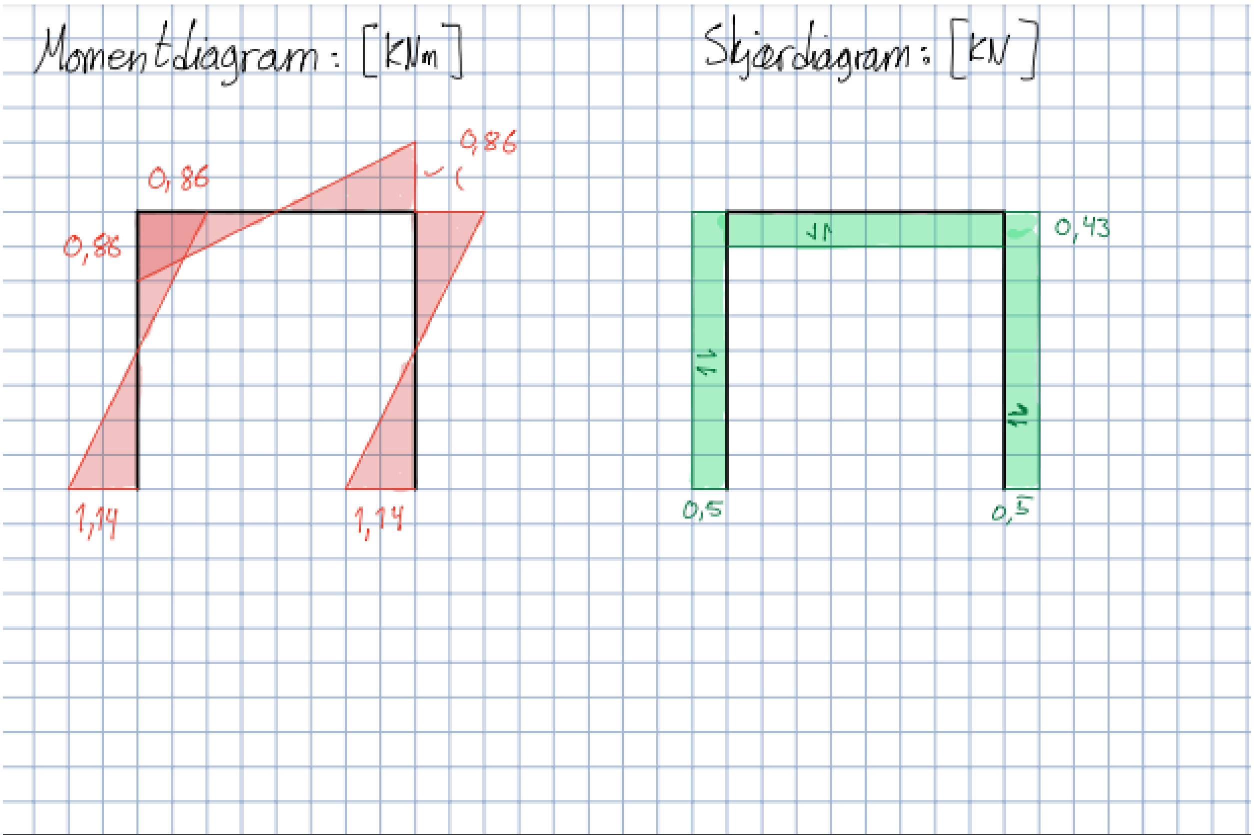
$$M_2 = \frac{6EI}{l^2}w_2 + \frac{4EI}{l}\theta_2 = \frac{6\cancel{EI}}{\cancel{l^2}}\cdot -\frac{5}{84}\frac{Pl^3}{\cancel{EI}} + \frac{4\cancel{EI}}{\cancel{l}}\cdot\frac{1}{28}\frac{Pl^2}{\cancel{EI}}$$

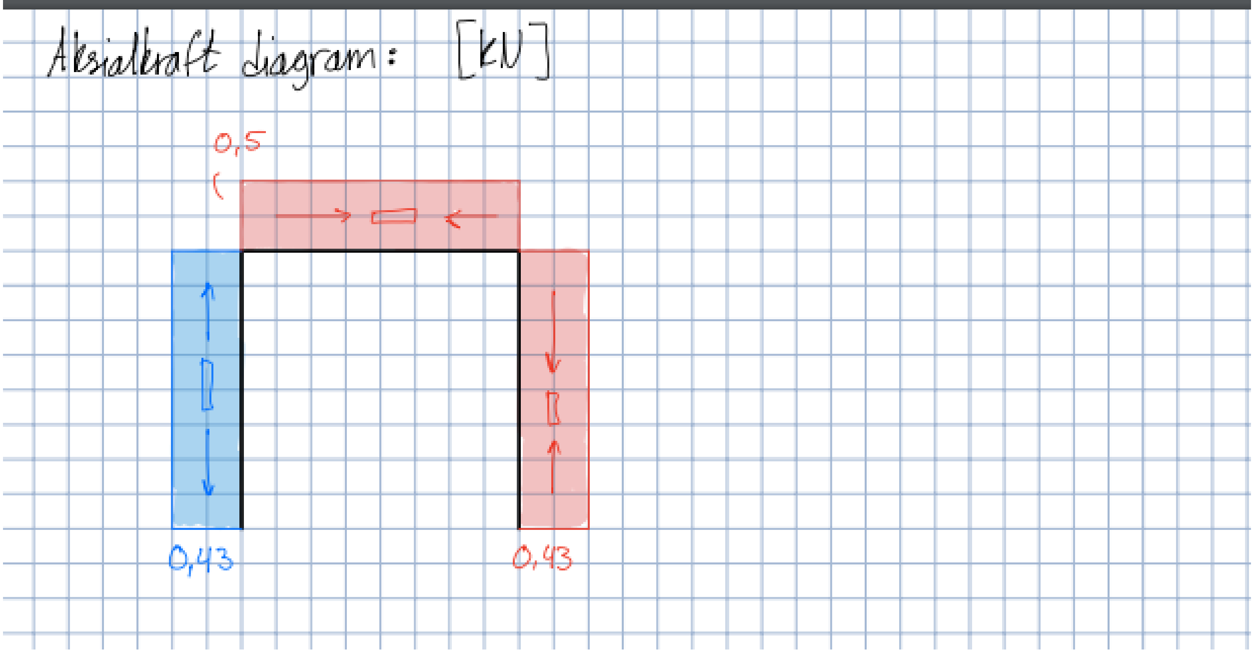
$$= 6\cdot\frac{-5}{84}Pl + \frac{1}{7}Pl = -\frac{5}{14}Pl + \frac{1}{7}Pl = -\frac{3}{14}Pl = -0{,}857\,\text{kNm}$$

$$Q_2 = \frac{12EI}{l^3}w_2 + \frac{6EI}{l^2}\theta_2 = \frac{12\cancel{EI}}{\cancel{l^3}}\cdot -\frac{5}{84}\frac{P\cancel{l^3}}{\cancel{EI}} + \frac{6\cancel{EI}}{\cancel{l^2}}\cdot\frac{1}{28}\frac{P\cancel{l^2}}{\cancel{EI}}$$

$$= -\frac{5}{7}P + \frac{3}{14}P = \underline{\underline{-\frac{1}{2}}}$$

portalramme 4 av 5





portalramme 5 av 5

E Diagrammer

E.1 Momentdiagram

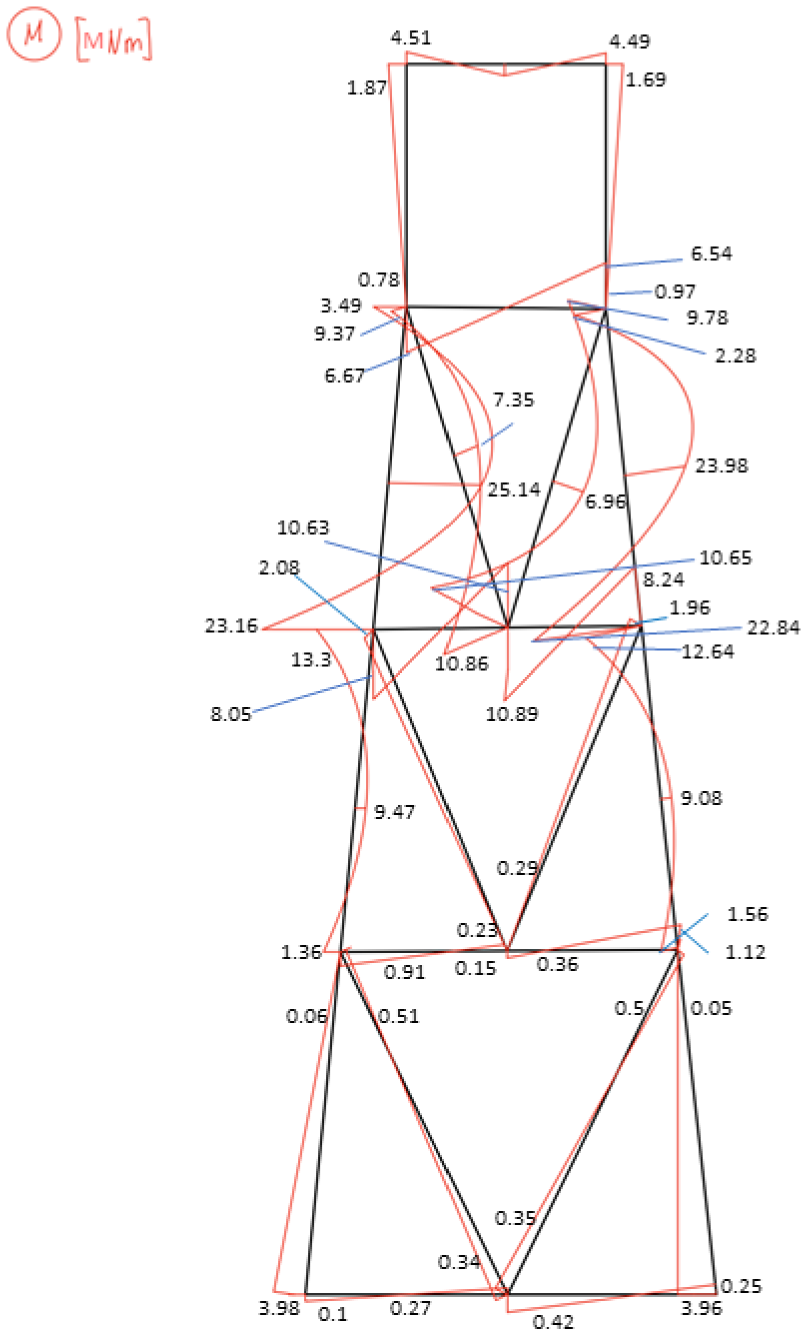


Figure 18: Momentdiagram fra Python-beregninger

E.2 Skjrkraftdiagram

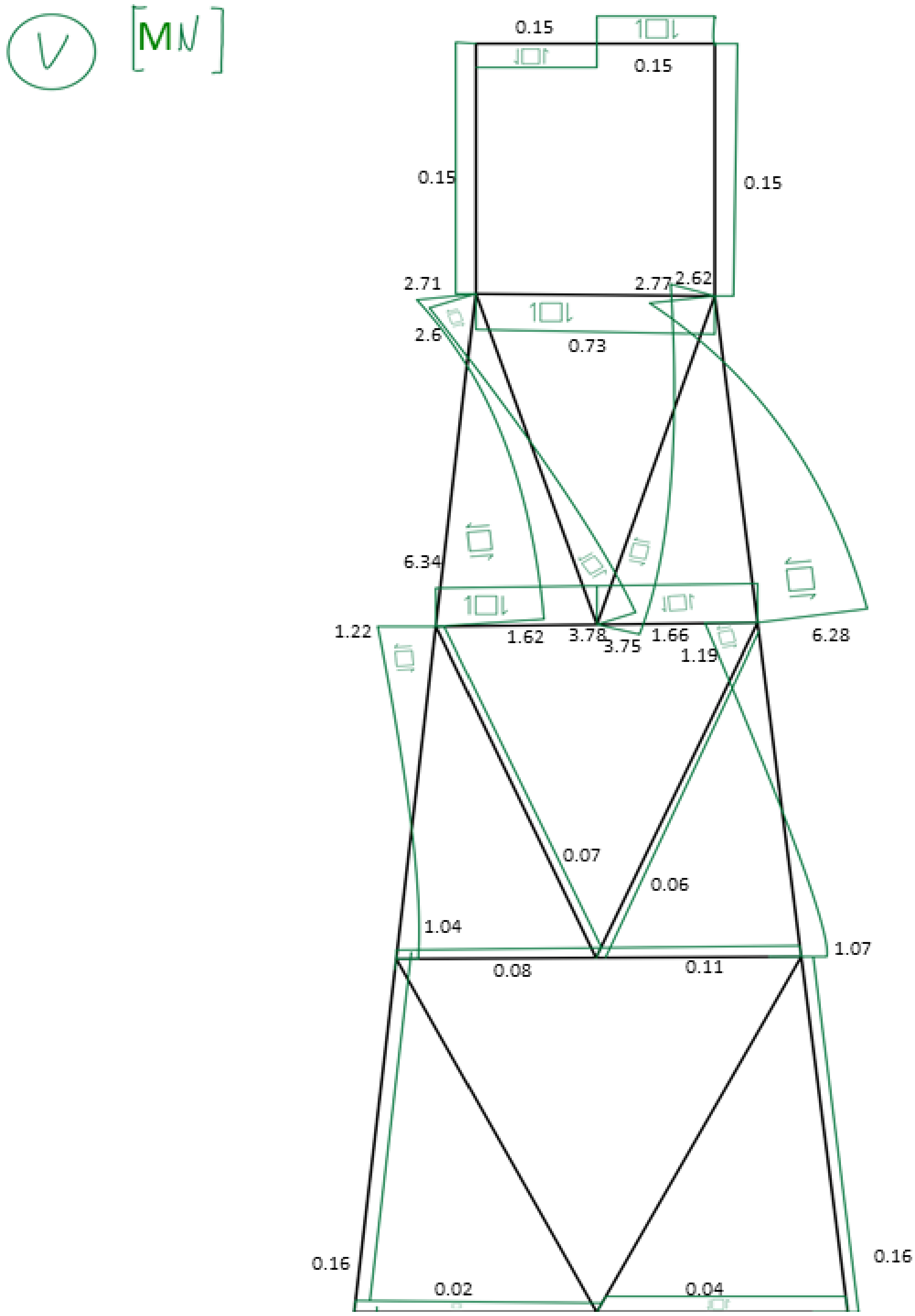


Figure 19: Skjaerkraftdiagram fra Python-beregninger

E.3 Aksialkraftdiagram

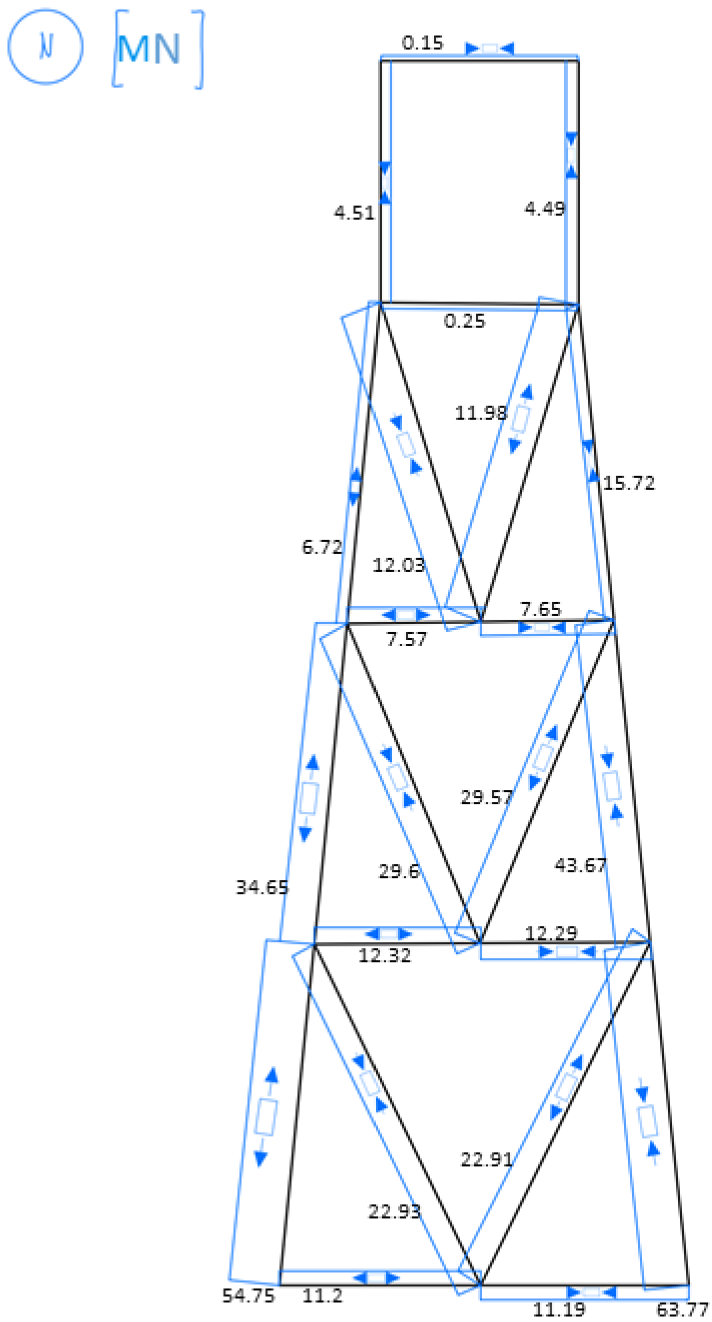


Figure 20: Aksialkraftdiagram fra Python-beregninger

E.4 Initialramme Python visualisering

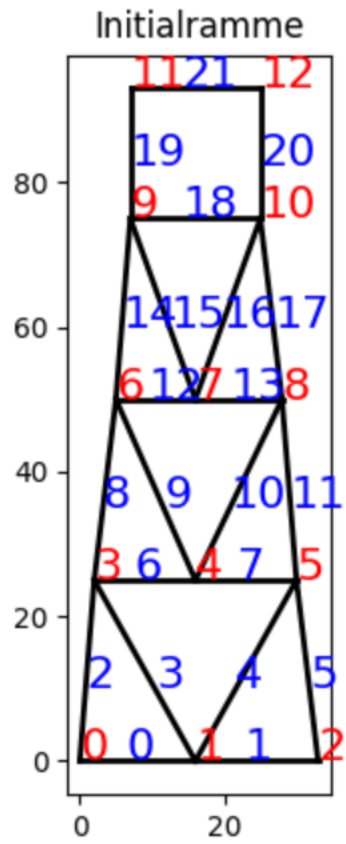


Figure 21: Initialramme Python visualisering

E.5 Deformert ramme Python visualisering

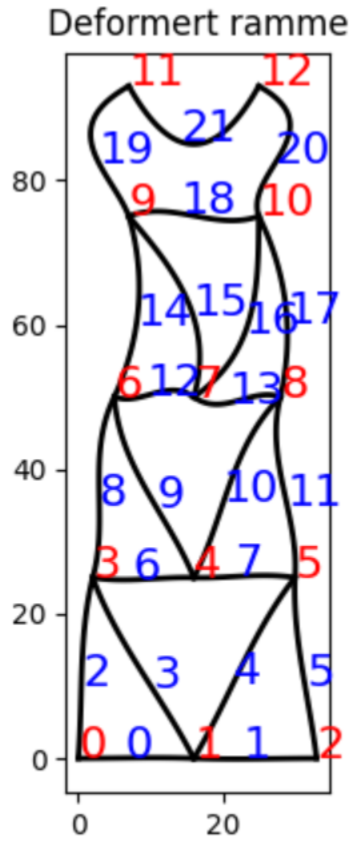


Figure 22: Deformert ramme Python visualisering

F Eksempelramme laster visualisert

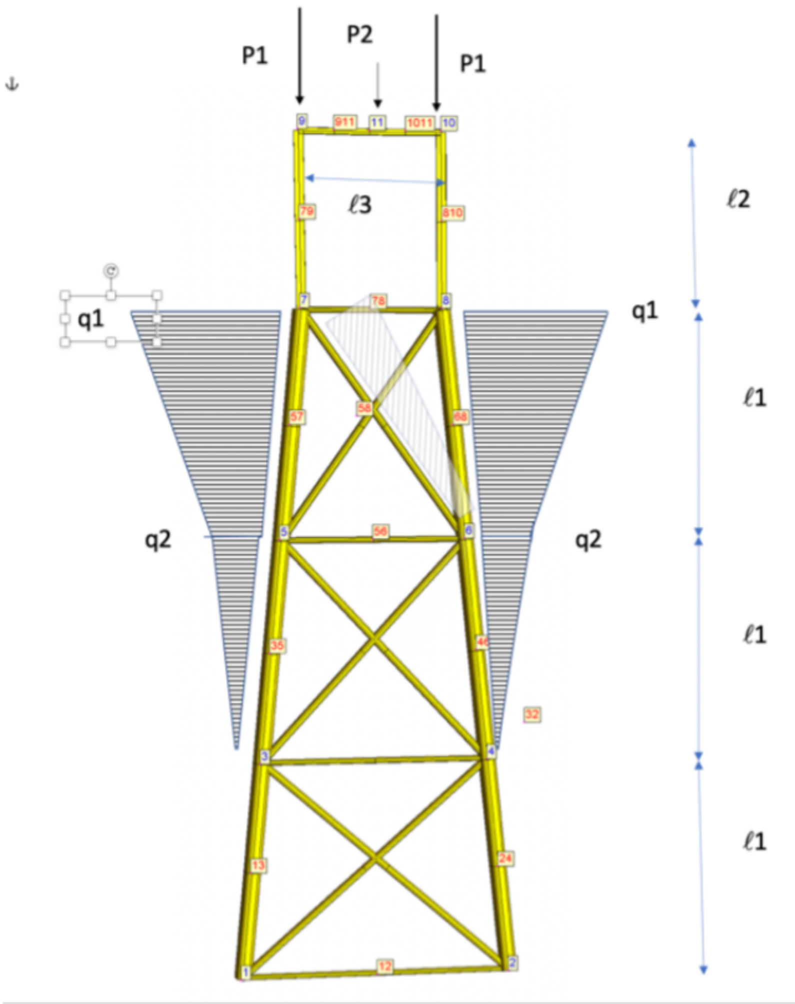


Figure 23: Eksempelramme laster visualisert