TMR4162 - Ramme Analyse

Simen Haugerud Granlund

30. november 2012

Sammendrag

Dette notatet er laget som en ledd av en innlevering i faget TMR4162 - Prosedyreorientert Programering i 2012. Det inneholder teori og informasjon om et program, kalt FEMA, som utfører en enkel styrkeberegning på en rammekonstruksjon. Programmet kan lastes ned via https://github.com/hgranlund/FEMA.

Simen Haugerud Granlund, Trondheim 2012.

Innhold

1 Innledning

Jeg valgte å ta oppgaven ramme analyse". Grunnen til dette er at jeg synes teorien rundt finite element method (FEM) er mye mer interessent og passer bedre til mitt studie enn Poisson-likningen. Jeg valgte også å legge til flere elementer i oppgaven. Oppgaven presiserte at vi bare trengte 2 frihetsgrader per bjelke, samt at vi kun trengte å presentere resultater for bøyemomentet. Jeg valgte å bruke 6 frihetsgrader per bjelke. Da fikk jeg verdier for både vertikal og horisontal forskyvninger, samt rotasjon. De resultatene jeg ville vise var; Moment diagram, skjærkraft diagram, Normalkraft diagram og starttilstanden.

Det jeg vil legge mest vekt må under oppgaven er god struktur, slik at jeg enkelt kan utvide med nye funksjonaliteter. DA er det også enkelt å sette seg inn i koden, man slipper å strukturere om mye på koden. Jeg liker generelt ikke mye kommentarer i koden. Koden skal være leselig og forståelig uten alt for mye kommentarer.

2 Teori

Her har jeg tenkt å kort beskrive de teoriene som er brukt i programmet. Jeg vil kun ta med de prinsippene, likningene og matrisene jeg brukte i koden og ikke beskrive i detalj hvordan jeg kan frem til dem basert på lærebøker.

2.1 Elementmetoden

Teorien bak elementmetoden ble utarbeidet på 1940-tallet, men man så ikke den store nytteverdien av den før datamaskinen kom til verden. Det første programmet som implementerte elementmetoden var NASTRAN skrevet i Fortran i 1965 [6]. Elementmetoden er i prinsippet en numerisk metode for å finne approksimasjoner til differensiallikninger. Den er brukt innenfor en rekke felter: strømning, varmeleding, svinginger, elektriske felt og liknende. I denne oppgaven bruker jeg den for å løse styrkebergnegrer på en rammekonstruksjon.

Elementmetoden går ut på å dele konstruksjonen inn i elementer, jo flere elementer jo mer nøyaktige resultater får man. Alle elementer har et vist antall noder. Selve rammekonstruksjonen blir representer av et nettverk av noder og elementer, hvor kreftene er festet til nodene. Det er i disse nodene verdier, som forskyving, krefter og momenter, blir kalkulert.

For hvert element blir stivheten til elementet kalkulert. Stivheten er basert på e-modulen, arealet og det andre arealmomentet til elementet. Disse stivhetene blir så addert sammen for å finne systemets stivhet. Vi kan deretter bruke sammenhangen mellom stivhet(k), forskyvninger(v) og krefter(S) (1) til å kalkulere knutepunktenes forskyvninger.

$$S = k * v \tag{1}$$

2.1.1 Lokal stivhetsmatrise

For å regne ut den lokale stivheten til et element kan en bruke stivhetsmatrisen (2).

$$k = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} & 0 & \frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} & 0 & \frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix} \tag{2}$$

Denne stivhetsmatrisen vil nå gjelde lokalt for det enkelte elementet. Vi må derfor transformere matrisen slik at den gjelder globalt. Dette kan vi gjøre med en enkel rotasjonsmatrise (3).

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\theta & -\sin\theta & 0 \\ 0 & 0 & 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Vi kan enkelt transformere kreftene S og forskyvningene v ved sammenhengene:

$$S_G = RS_L \ , \ v_G = Rv_L \tag{4}$$

Vi kan nå ved hjelp av 1 og 4 finne en sammenheng mellom den globale $_G$ og den lokale $_L$ stivhetsmatrisen.

$$\begin{aligned} S_G &= RS_L = Rk_Lv_L = Rk_LR^{-1}v_G = k_Gv_G \\ k_G &= Rk_LR^{-1} \end{aligned} \tag{5}$$

2.1.2 Global stivhetsmatrise

Om vi studerer den lokale stivhetsmatrisen 2 og likning 1 ser vi at kolonne nummer i beskriver stivheten til frihetsgraden i. Dette er også tilfelle for den globale stivhetsmatrisen. Den globale stivhetsmatrisen holder på stivhetene til alle nodene i systemet, mens den lokale stivhetsmatrise inneholder kun stivheten til sine tilhørende noder. Når de adderes, skal de lokale stivhetene adderes inn i de tilhørende stivhetene i den globale stivhetsmatrisen. Normalt brukes en IEG-matrise, som beskriver systemets topografi, til å addere stivhetene sammen. IEG-matrisen er en relasjon mellom elementene og det tilhørende nodeparret.

Vi kan lett kalkulere størrelsen på den globale stivhetsmatrisene. Det første en bør merke seg er at alle stivhetsmatriser er symmetriske, noe som kan bevises ved Maxwells resiprositetssats. Dette her også en følge av newtons 3. lov, siden hver enkelt bjelke må være i likevekt. Det andre er at hver kolonne representerer en frihetsgrad. For å finne størrelsen på matrisen må en derfor beregne antall frihetsgrader til systemet.

2.1.3 Grensebetingelser

Nå står vi bare igjen med et lineært ligningssystem av typen $kv = S$, hvor forskyvningene (v) er ukjente. Dette likningssystemet vil nå være uløselig, fordi stivhetsmatrisen k er singulær. Denne singulariteten skyldes at vi enda ikke har sakt hvor konstruksjonen er fastbundet, vi har ingen

grensebetingelser. For et system med 3 frihetsgrader per node trenger vi 3 fastlåste frihetsgrader. Disse frihetsgradene kan innføres på to måter. Enten kan raden og kolonnen til frihetsgraden fjernes, ellers kan alle verdiene nulles ut bortsett fra diagonal-verdien som må være en. Når vi har innført disse grensebetingelsene vil systemet være løselig.

2.1.4 Elementkrefter

Når vi vil regne ut krefter på de forskjellige elementene kan vi igjen bruke ligning 1. Vi må nå huske at vi må transformere matrisene til det koordinatsystemet vi ønsker.

2.2 Ligningsløser

Når man skal modellere mange fysiske og interiørmessigere problemer kan en nesten ikke unngå å løse store linear ligningssystem. Det finnes derfor mange forskjellige metoder å løse disse på og alle har visse fordeler og ulemper. Det finnes to kategorier av ligningsløsere: dirkete og iterative. De direkte løser settet eksakt, men bruker ofte lenger tid en en iterativ løser. Gauss metoden er et eksempel på en direkteløser. En iterativ løser finner en approksimasjon av løsningen, og kan ha problemer med konvergens i visse tilfeller.

2.2.1 Gauss

Gauss metoden består av to deler. Den ene er eliminasjon, hvor du utfører elementære rad operasjoner til du for en triangulær form. Deretter gjøres en tilbakesubstitusjon for å finne x-verdiene. De finnes i hovedsak 3 forskjellige rad operasjoner:

1. Multiplisere en rad med et tall som ikke er null.
2. Addere to rader
3. Bytte om to rader

Kjøretiden til gauss kan kalkuleres ut i fra antall iterasjoner som gjøres. Om vi antar det er n likninger som skal løses kan kjøretiden kalkuleres som vist i (6).

$$E(n) = \sum_{k=1}^{n-1}(n-k) + 2\sum_{k=1}^{n-1}(n-k)(n-k-1) = O(n^3) \tag{6}$$

På samme måte kan vi kalkulere kjøretiden til tilbakesubstitusjonen:

$$T(n) = 2\sum_{i=1}^{n}(n-i) + n = O(n^2) \tag{7}$$

2.2.2 Gauss-Seidel

Gauss-seidel er en iterativ ligningsløser, som fungerer best på spinkle matriser. Den har en iterativ del, beskrevet i likning **??**. I hver iterasjon bruker Gauss-Seidel den forrige utregnede verdien, som gjør at den ikke kan kjøre parallelt (på normalt vis).

$$x_j^{m+1} = \frac{1}{a_{jj}}(b_j - \sum_{k=1}^{j-1} a_{jk}x_k^{m+1} - \sum_{k=j+1}^{n} a_{jk}x^m) \tag{8}$$

Her kan vi ikke finne kjøretiden på samme måte som ved en direkteløser, siden vi hele tiden kan forbedre resultatet ved å kjøre en iterasjon til. Vi må derfor se på konvergens, parallellitet og hvilken nøyaktighet man skal ha. Det finnes noen enkle konvergeringsregler for Gauss-Seidel, som sier at metoden garantert konvergerer om:

1. Matrisen (A) er strengtsasssm eller uavvendelige diagonalt dominerende.
2. Matrisen (A) er positive bestemt (Positive-definite)

2.3 OpenGL

Opengl er et API for å rendre 2d og 3d grafikk. Mange av kallene går rett på GPU'en, slik at vi får økt ytelsen. Strukturen er bygget opp som en tilstandsmaskin, hvor vi hele tiden forandrer tilstanden til programmet. Vi kan for eksempel sette hvilken farge eller linjetykkelse som skal brukes videre, ved kommandoene glColor3f() og glLineWith().

OpenGL opererer med en stakk av 3 matriser, hvor en på hver matrise kan utføre operasjoner som rotasjoner, forskyvninger og skaleringer. Hver gang en matrise operasjon blir kalt blir den utført på den gjeldene matrisen og matrisene lenger nede i stakken blir ikke påvirket. De forskjellige operasjonene er standard matrise operasjoner, hvor matrisen blir multiplisert med en operasjons matrise.

$$glRotate(\theta,x,y,z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

$$glScale(x,y,z) = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

$$glTranslate(x,y,z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

For å lage og dytte ut matriser på stakken brukes glPushMatrix() og glPopMatrix(). Når du har hentet ut en matrise kan du utføre rutiner på denne. Modellmatrisen er den mest essensielle matrisen, her kan du utføre rutiner som å tegne streker, polygoner og lignende.

3 Programmet

3.1 Beskrivelse

Dette programmet utfører en enkel styrkeberegning på en en rammekonstruksjon ved hjelp av elementmetoden. Her blir hver bjelke omgjort til et element med 6 frihetsgrader. Analysen kan ta imot krefter eller momenter påsatt i nodene

Programmet er bygget opp av to deler. Det første er en beregnende del, mens den andre er en visualiserende del. Beregningsdelen, programmet FEM, er skrevet i Fortran 90. FEM leser

en input fil og utfører en elementanalyse. Resultatene av elementanalysen blir så skrevet til FEMOutput.dat. Denne filen igjen lest av det visualiserende programmet, kalt FEMvis. Denne delen er skrevet i c og viser frem resultatene.

3.1.1 Installasjon

For å bygge programmet brukes kommandoene:

```
cd FEMA/src/;                Gå til mappen FEMA/src.
make;                        Bygger både Fortran og c code.
```

3.1.2 Bruk

Jeg har laget to bash script til å kjøre programmet, som heter run og runWithInput. Disse finnes projectes root-mappe. Run trenger ingen input fil, men bruker filen */inputFiles/input.dat* som standard input. Det det andre skripet krever en input fil. En del ferdiglagde input filer finnes i mappen *inputFiles/*. Eksempler på bruk:

```
./run;
```

```
./runWithInput < inputFiles/input2.dat
```

Når kommandoene kjøres vil et resultatvindu åpnes. Dette vinduet kan kontrolleres med tastene:

'I' - Visualiserer inputen.
'F' - Viser Rammen.
'M' - Viser momentdiagrammer.
'S' - Viser Skjærkraft diagrammer
'A' - Viser Aksialkraft diagrammer.

3.1.3 Input filen

Input filen beskriver systemet som skal beregnes. Her skal det spesifiseres hvilke noder, elementer og laster som skal være med i kalkulasjonen. Hvilken benevning som brukes er valgfritt så lenge den er konsekvent. Input filen blir lest igjennom standard input enheten. ur 1 viser et eksempel på en input fil.

3.2 Fortran

Delen som er skrevet i Fortran implementerer en enkel styrkeberegning basert på elementmetoden. Her er det blant annet også implementert en Ligningsløser som baserer seg på gauss eliminasjon. I Fortran delen har jeg hentet mye info om oppbygging og strukturering fra Kolbein Bell sine notater [5] [4].

3.2.1 Struktur

Jeg har valgt å strukturere programmet inn i 5 moduler og et hovedprogram. Modulene og hovedprogrammet er delt inn følgende:

FEM er navnet på hovedprogrammet mitt. Det er dette programmet som leser og skriver data til fil og kaller på metoder i modulen FEMMetods.

```
6,6,1                   ! AntallNoder, AntallElementer, AntallKrefter
0,0,0,0,0               ! Liste med noder, på format:
0,9000,1,1,1            ! x-verdi, y-verdi, frihet i x-retning, frihet i y-retning, rotasjonsfrihet
6000,9000,1,1,1
9000,9000,1,1,1
9000,6000,1,1,1
9000,0,0,0,0,0
200E3,6500,80E6,2,3     ! Liste med elementer, på format:
200E3,6500,80E6,3,4     ! E-modul,Areal, I, node1, node2
200E3,6500,80E6,3,5
200E3,6500,80E6,4,5
200E3,6500,80E6,5,6
3,2,-40000              ! Liste med krefter,på format:
                        ! NodeNummer, Hvilken frihetsgrad? (x=1,y=2,r=3), Value
```

Figur 1: Eksempel på en input fil

FEMMethods er hovedmodulen. Dette er den eneste modulen hovedprogrammet bruker. Her finnes alle metoder som har med finite element method"å gjøre. Hovedrutinen, som utfører styrkeberegningen, heter DoFEM.

FEMUtility er en hjelpemodul for FEMMethods. Her finnes mange støtterutiner for de mer essensielle rutinene som finnes i FEMMethods. Her er rutiner som for eksempel nullstiler eller printer en matrise.

FEMTypes er en modul som inneholder de forskjellige datatypene som brukes. De datatypene som brukes er element, joint og load.

FEMMath er mattemodulen. Her finnes alle rutiner innenfor matematikk som for eksempel rutiner for å utføre Gauss eliminasjon, Gauss-Seidel eller generere en rotasjonsmatrise.

Jeg har strukturert modulene og programmet som vist i figur 2. Her ser vi at det er hovedprogrammet som tar for seg lesing og skriving til fil og kaller rutinen DoFEM i modulen FEMMethods. DoFEM gjør da de nødvendige kallene for å kalkulere krefter, momenter og forskyvninger på alle elementene. I figuren har jeg ikke tatt med kallene til rutiner som er ubetydelige for forståelsen en strukturen.

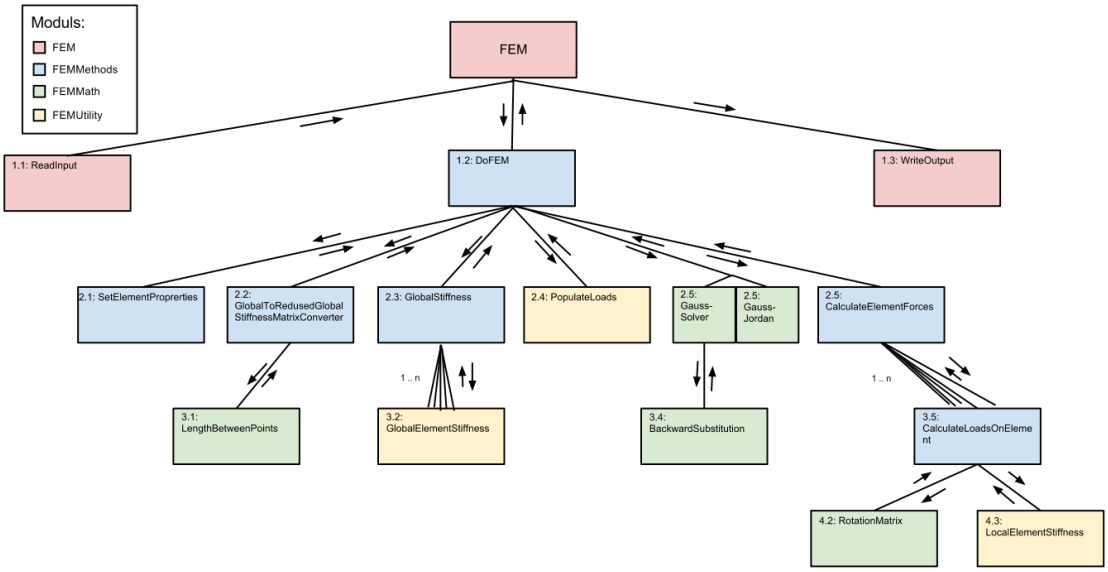
Datastrukturen jeg har brukt for å representere rammekonstruksjonen og elementene er ved hjelp av egendefinerte datatyper. Den infoen som må holdes på er data om noder, elementer og krefter. Dette har jeg valgt å strukturere ved hjelp av tre datatyper:

Joint inneholder info om plasseringen til noden, x og y koordinater, og hvilke frihetsgrader noden har.

Element består av info som tilhører de enkelte elementene som blant annet E-modul, areal , rotasjon og hvilke noder den er koblet til.

Load inneholder hvilken node og retning kraften har og størrelsen på kraften.

Disse datatypene blir så lagret i tre lister, som blir modifisert og brukt igjennom hele program flyten. Selve programflyten er vist i figure 3.



Figur 2: Program strukturen

Her vises også feilhåndterings-flyten i programmet. Dette er implementert som anvist i Kolbein [4], med et errorflagg.Flagget brukes slik:

$$error flag = 0 : ingen feilsituasjoner på vist$$
$$error flag < 0 : en feilsituasjoner på vist$$
$$error flag > 0 : en tvilsom situasjoner på vist$$

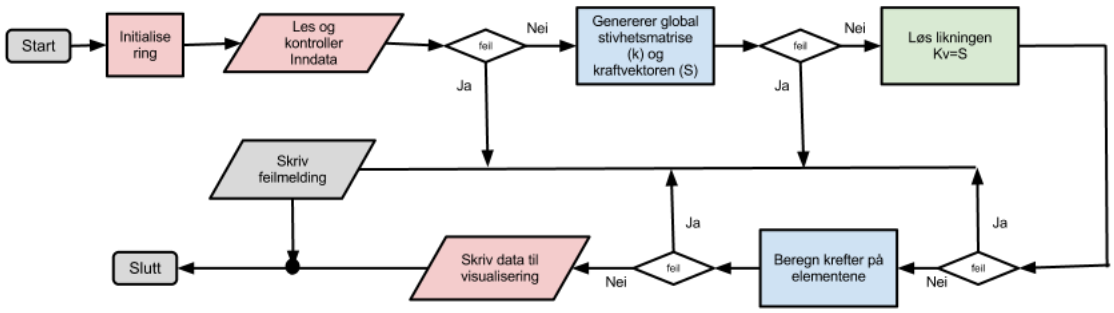
Et annet element jeg fant veldig nyttig fra Kolbein [4] var ideen bak variabelen pr_switch. Ideen er at print svitsjen har en verdi mellom 0 og 10, hvor 10 printer alt og 0 ikke printer noe. Dette fant jeg veldig nyttig når jeg skulle feilsøke koden. Tanken er at print svitsjene aldri skal bli fjernet slit at det skal bli lett å feilsøke koden ved en senere anledning.

3.2.2 Diskusjon rundet Optimaliseringer og tidsbruk

Når vi skal se på den asymptotiske kjøretiden til programmet, er det 2 algoritmer vi må tenke på. Den ene er ligningsløseren, mens den andre er generering av den globale stivhetsmatrisen. Om vi angir n til å være grader av frihet i kunstaksjonen og e til å være antall elementer, kan vi kalkulere kjøretiden. Kjøretiden til Gauss eliminasjonen er vist i (6). For stivhetsmatrisen blir alle elementene iterert over og For hvert element addert inn i hovedmatrisen. Kjøretiden blir derfor:

$$T(n) = \sum_{k=1}^{e}(k) * (konstant) \sum_{j=1}^{n}(j)(j)(konstant) = O(e * n^2) \tag{12}$$

Den første enkle justeringen jeg gjorde var å kalkulere den globale elementstivhetsmatrisen direkte. Vanligvis ville denne matrisen blitt laget ved å bruke likning 2 og 3 på formen $k_G =$



Figur 3: Program flyten

$R * k_L * R^T$. Det vil si at vi trenger å gjøre to matrisemultiplikasjoner og en transponering, for å få rotert matrisen. Jeg valgte da å kalkulere denne, på forhånd, slik at k_G blir.

$$k_{globalElementStivhet} = \frac{E}{L} * \begin{bmatrix} AC^2 + \frac{12I}{L^2}S^2 & (A - \frac{12I}{L^2})CS & -\frac{6I}{L}S & -(AC^2 + \frac{12I}{L^2}S^2) & -(A - \frac{12I}{L^2})CS & -\frac{6I}{L}S \\ & AS^2 + \frac{12I}{L^2}C^2 & -\frac{6I}{L}S & -(A - \frac{12I}{L^2})CS & -(AS^2 + \frac{12I}{L^2}C^2) & -\frac{6I}{L}S \\ & & fI & \frac{6I}{L}S & \frac{6I}{L}C & 2I \\ & & & (AC^2 + \frac{12I}{L^2}S^2) & (A - \frac{12I}{L^2})CS & \frac{6I}{L}S \\ & & & & (AC^2 + \frac{12I}{L^2}C^2) & -\frac{6I}{L}C \\ Symmetri & & & & & 4I \end{bmatrix} \quad (13)$$

$\texttt{Where} : \texttt{E} = \texttt{E} - \texttt{modul}, \texttt{I} = \texttt{Annetarealmoment}, C = cos(\theta), S = \sin(\theta)$

Dette vil ikke forbedre kjøretiden asymptotisk sett, men vil fjerne noen konstantledd.

En annen ting jeg gjorde var å forbedre genereringen av den globale stivhetsmatrisen. Dette blir vanligvis gjort ved å først lage den fulle matrisen deretter, ved hjelp av grensebetingelser, redusere matrisen. For å unngå to operasjoner lagde jeg den reduserte matrisen direkte. Dette gjorde jeg ved å lage en konverteringsvektor, som mappet verdier fra den tenkte fulle til den reduserte globale stivhetsmatrisen. Jeg laget også en konverteringsvektor fra de de enkelte elementstivhetsmatrisene til den globale stivhetsmatrisen. Dette gjorde at jeg ikke trengte å iterere igjennom hele den globale stivhetsmatrisen hver gang en elementmatrise skulle adderes. Dette vil si at kjøretiden på å lage den globale stivhetsmatrisen gikk fra (12) til(14)

$$T(n) = \sum_{k=1}^{e} (k) * (konstant) = O(e) \quad (14)$$

Det område jeg muligens kunne spare mest tid var å velge en god iterativ ligningsløser. Jeg lagde derfor en implementasjon av gauss-seidel. Teoretisk sett er det lurest å gå for en iterativ ligningsløser som Gauss-Seidel. Grunnen til dette er at stivhetsmatrisen som oftest vil være spinkel, siden de fleste frihetsgradene ikke er avhengig av hverandre. De enkelte nodene er kun avhengig av de nærliggende nodene som gjør at vi, som oftest, vil få mage null-verdier i matrisen.

Vi vil også få en sterk og alltid utfylt diagonal. Dette er to elementer som er svært positivt for konvergeringen til Gauss-Seidel. Det vil allikevel lønne seg å bruke Gauss eliminasjon på små matriser. Programmet vil derfor, avhengig av størrelsen på stivhetsmatrisen, velge hvilken ligningsløser som er best. Den avgjørende verdien hvor Gauss-Seidel tar igjen Gauss eliminasjon har jeg ikke hatt mulighet til å se på enda. For å gjøre en slik analyse ville jeg vært nøtt til å få et større datasett. Dette kunne vært gjort ved å meshe opp konstruksjonen i mindre deler. I mitt program kunne dette gjøres ved å innføre et vist antall noder, med 3 frihetsgrader, i hver av bjelkene.

En annen ting som er vært å peke på når det gjelder Fortran, er minnehåndtering. Her kan man for eksempel se på hvordan matriser er lagret. Fortran lagrer matriser i en minneblokk med kolonnene etterhandene. Det vil derfor være lurt å traversere kolonne for kolonne. Med tanke på minnehåndtering har jeg valgt å lese hele input filen inn i minne instedet får å lese/skrive til en fil under programutføringen. Dette gjør at koden blir hurtigere siden vi ikke trenger å lese/skrive fra disk, men kan lese rett fra ram. Ulempen med dette er, om dataen skulle bli enormt stor, at vi bruker for mye minne. Denne risken har jeg valgt å utelate med tanke på dagens datamaskiner har store mengder med minne. Jeg har heler valgt å bruke deallocate på data jeg vet jeg ikke får bruk for lenger.

3.2.3 Resultater

De resultatene Fortran produserer er verdier på hver enkelt node. Disse verdiene er:

- Forskyvning i i x og y-retning.
- Rotasjon om punktet.
- Kraft i x og y-retning.
- Moment om noen.

Ut i fra disse verdiene kan moment, skjær og normalkraft diagrammer bli kalkulert. Rammens plassering etter kraft påføring kan også bli vist. Det Fortran skiver i outpufilen er ikke ment å være leselig for mennesker, men er brukt for å lette arbeidet til visualiseringsprogrammet. Det som blir skrevet er:

- Hjelpevariable som; antall elementer og laster, samt den største koordinatverdien.
- En liste med alle node koordinatene til elementene.
- En liste med kreftene og momentene til elementene
- En liste med forskyvningene til elementene.
- En liste med de påførte kreftene.

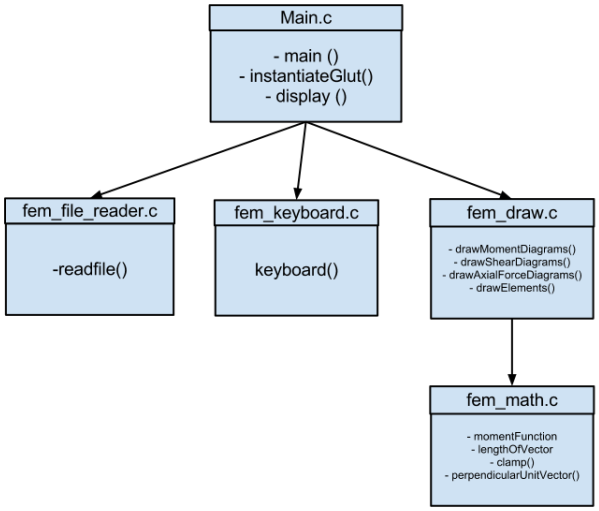
Basert på disse verdiene kan resultatene bli visualisert.

3.3 C

Denne delen av programmet visualiserer resultantene fra Fortran med en enkel brukerintetaksjon. Dette blir implementert ved hjelp av OpenGL og biblioteket glut.

3.3.1 Struktur

Selve program-flyten er bygget opp etter glut, hvor du har en mainloop som kontrollerer flyten. Her blir alt av tegning, tastatur input og omforming av vinduet kontrollert. Når jeg programmerer liker jeg å ha programmet delt opp, så jeg har også her valgt å strukturere koden i flere enheter. Sammenhangen mellom enhetene og de viktigste rutinene er vist i figur 4.



Figur 4: En oversikt over sammenhengen mellom enhetene og de tilhørende hovedfunksjonene i C

FEMVis.c: Dette er hovedenheten og inneholder main metoden. Her initialiseres glut ved å sette rutiner som kaller ved tastatur input, omforming av vindu, tegning og liknende.

fem_file_reader.c: Her leses input filen.

keyboard: Denne enhten inneholder funksjonaliteten som styrer tastatur input.

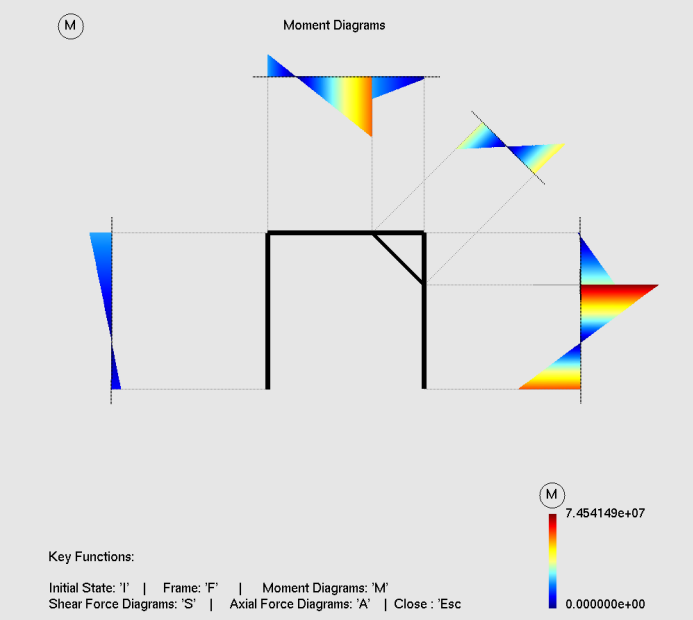
fem_draw.c: Enheten inneholder alle tegnefunksjoner. Hovedfunksjonene er her tegning av diagrammene og bjelkene. Disse kaller igjen videre på mindre spesifikke rutiner som tegning av text, piler osv.

fem_math.c: Dette er matte enheten, og inneholder alle rutiner som har med matematikk å gjøre.

Et annet element i struktureringen av et c program er header filen som gjør koden mer oversiktlig å lesbar. Header filen kan sammenliknes med et interface fra objektorientert programmering. Her kan det defineres variable, rutiner, funksjoner og liknende. Denne filen blir lest av preprosessoren og gjør at kompilatoren for mer informasjon om deklarasjonene i filen. Fordelene med dette er at vi ikke trenger å inkludere kildekode, men bare header filen for å kunne bruke biblioteker.

3.3.2 Visualisering

Visualiseringen av elementanalysen har flere tilstander som styres av tastaturkommandoer. Disse tilstandene er visning av input, rammen, moment diagram, skjærkraft diagram og aksialkraft diagram. Visningen av momentdiagrammene vises i figur 5. Her vises også navigasjonsmenyen nederst i venstre hjørne.



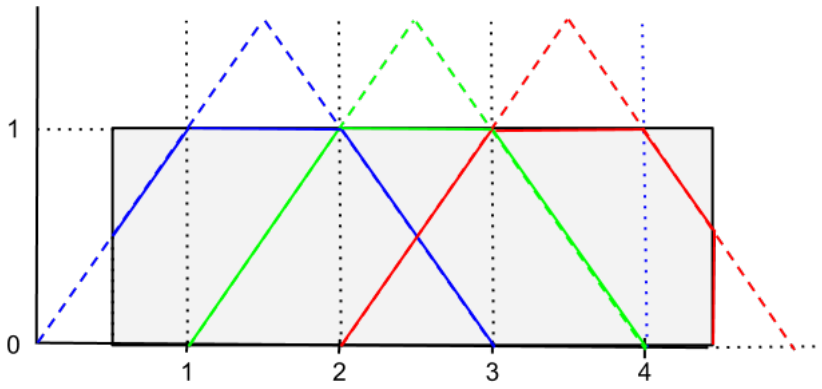
Figur 5: Visualiseringen av momentdiagramemne fra C

For å representere diagrammene valgte jeg å bruke en fargefunksjon og en fargeindikator. En fargefunksjon er en funksjon genererer en farge basert på en verdi, hvor min returnerte en rgb-verdi. Fargemodellen rgb baserer seg på og blande, blå, grønn og rød, til ønsket farge. På figur 6 ser vi hvordan min farge funksjon varierer. Måten jeg gjorde dette på var å først lage en rgb-verdi basert på de stiplede linjene, forså å klemme verdiene inn i den grå boksen.

4 Konklusjon

I løpet av dette prosjektet har jeg lært mye om prosedyreorientert programmering og språkene Fortran og C. Ved prosedyreorientert programmering var det et nytt tankesett som var den største forandringen. Før var jeg vant til å programmere objektorientert, hvor tenkesette gitt ut på å dele koden opp i objekter og gjøre operasjoner på diss. Tankesette ved prosedyreorientert programmering går mer på pipe et datasett gjennom forskjellige rutiner for å oppnå ønsket resultat. Etter at jeg nå har blitt kjent med prosedyreorienter programmering ser jeg at begge verdenene har sine fordeler og ulemper.

Tankesettet er også anderledes når du programmerer i språk av lavere orden enn det jeg vanligvis her programmert. Nå tenker man ikke lenger på datastrukturer som hash-map, sett, eller lenkede lister, men på minneblokker og pekere. For dette prosjektet tenkte jeg ikke på en matrise



Figur 6: Dette er fargefunksjonen. Den grå boksen viser funksjonens gyldighetsområde.

eller vektor som en minneblokk med en innebygget aksesseringsfunkjson. De nye tankemønstretere tror jeg kommer godt med i videre programmering. Jeg vil ha en bedre forståelse av hva som skjer på minne og bit nivå.

Avslutningsvis vil jeg si at jeg lærte mye under dette prosjektet. Jeg likte veldig godt den frie oppgavestilen som gjorde at jeg kunne velge noe jeg fant motiverende og som passet til mine studier. Det er også mye læring i praktisk arbeid og nå føler jeg at jeg har en vis innsikt i prosedyreorienterte verden og språkene Fortran og C

Referanser

[1] Åge Ø. Waløen, 1995, Dimensjonering ved hjelp av elementmetoden, NTNU

[2] Dyryl L. Logen, 2011, A First Course in the Finite Element Method

[3] Erwin Kreyszig, 2006, Advanced Engineering Mathematics, 9th Edition

[4] Kolbein Bell, 2007, Programutvikling (Teknisk, prosedyreorientert programmering), NTNU

[5] Kolbein Bell, 2006, Fortran 90, NTNU

[6] Wikipedia, http://en.wikipedia.org/wiki/Finite_element_method