NTNU
Kunnskap for en bedre verden

Utforming av skallkonstruksjoner

Roger Konnestad

Master i Bygg- og miljøteknikk
Innlevert: juni 2018
Hovedveileder: Anders Rönnquist, KT
Medveileder: Marcin Luczkowski, KT

Norges teknisk-naturvitenskapelige universitet
Institutt for konstruksjonsteknikk

Institutt for konstruksjonsteknikk
Fakultet for ingeniørvitenskap
NTNU- Norges teknisk- naturvitenskapelige universitet

TILGJENGELIGHET

ÅPEN

MASTEROPPGAVE 2018

FAGOMRÅDE: Ingeniørarkitektur	DATO: 11.06.2018	ANTALL SIDER: 68

TITTEL:

Utforming av skallkonstruksjoner

Design of Shell Structures

UTFØRT AV:

Roger Konnestad

SAMMENDRAG:
Skallkonstruksjoner er en gammel form for byggverk med lange tradisjoner. De er spektakulære byggverk som strekker seg over lange spenn og kan bli stående i flere tusen år. Hvordan man utformer slike konstruksjoner har forandret seg opp igjennom historien. I starten var det kun ved hjelp av fysiske tester man kunne dimensjonere og utforme disse konstruksjonstypene. I dag kan alt dette gjøres ved hjelp av digitale verktøy, som gir ytterligere muligheter for optimalisering av konstruksjonene.
Det finnes i dag flere numeriske metoder som kan brukes til å finne form på skallkonstruksjoner. I denne oppgaven har noen av disse metodene for formfinning blitt studert, sammenliknet og implementert i parametriske modelleringsverktøy for å utforme skallkonstruksjoner. De forskjellige metodene er testet og analysert for å se hvilken nytte de har, og hvor gode resultater de gir.
Ved å bruke disse metodene i parametriske verktøy åpner det opp for nye måter å utforme og planlegge skallkonstruksjoner. Det å finne nye former går raskt, og modellene kan analyseres med en gang. Endringer kan gjøres uten å måtte gjøre alt på nytt igjen. Dette betyr at store deler av designprosessen kan gjøres i et program der alle leddene henger sammen.

FAGLÆRER: Anders Rønnquist

VEILEDER(E): Anders Rønnquist, Marcin Luczkowski

UTFØRT VED: Institutt for konstruksjonsteknikk

Sammendrag

Skallkonstruksjoner er en gammel form for byggverk med lange tradisjoner. De er spektakulære byggverk som strekker seg over lange spenn og kan bli stående i flere tusen år. Hvordan man utformer slike konstruksjoner har forandret seg opp igjennom historien. I starten var det kun ved hjelp av fysiske tester man kunne dimensjonere og utforme disse konstruksjonstypene. I dag kan alt dette gjøres ved hjelp av digitale verktøy, som gir ytterligere muligheter for optimalisering av konstruksjonene.

Det finnes i dag flere numeriske metoder som kan brukes til å finne form på skallkonstruksjoner. I denne oppgaven har noen av disse metodene for formfinning blitt studert, sammenliknet og implementert i parametriske modelleringsverktøy for å utforme skallkonstruksjoner. De forskjellige metodene er testet og analysert for å se hvilken nytte de har, og hvor gode resultater de gir.

Ved å bruke disse metodene i parametriske verktøy åpner det opp for nye måter å utforme og planlegge skallkonstruksjoner. Det å finne nye former går raskt, og modellene kan analyseres med en gang. Endringer kan gjøres uten å måtte gjøre alt på nytt igjen. Dette betyr at store deler av designprosessen kan gjøres i et program der alle leddene henger sammen.

Abstract

Shell structures is an old type of construction with long traditions. They are spectacular buildings who stretches over long spans and stands for thousands of years. How we form these kinds of constructions have went through an evolution up through the history. From physical testing being the only way to design these types of structures to today when everything can be done by the help of digital tools.

There are today several numerical methods that can be used for form-finding of shell structures. In this thesis some of these form-finding methods have been studied, compared and implemented in parametric modeling tools to design shell structures. The methods have been tested and analyzed to find their use and to see how good results they give.

By using these methods in parametric tools whole new ways of designing and planning of shell structures emerges. One can quickly find new forms and the models can be analyzed immediately. Changes can be done without having to do all previous work again. This means that a large part of the design prosses can be done in one program where all the parts are connected.

Forord

Denne masteroppgaven ble skrevet våren 2018 ved Norges teknisk naturvitenskaplige universitet, som avslutning på mastergraden i Bygg og Miljøteknikk, ved Institutt for konstruksjonsteknikk under temaområdet ingeniørarkitektur.

Denne oppgaven har for meg vært spennende og utfordrende og jeg har lært mye. Å modellere og formgi har alltid vært en av mine store interesser, og det å få mulighet til å jobbe mer med dette har vært veldig interessant og givende. Spesielt spennende har det vært å programmere de forskjellige metodene for derigjennom å komme frem til en form som har blitt anvendt inn i et annet modelleringsprogram og få alt til å fungere sammen.. Det å programmere er noe jeg har hatt liten erfaring med fra tidligere, men alltid ønsket å lære mer om. Nesten all programvare som er brukt i denne oppgaven var nye for meg, men etter en god start med noe opplæring, og deretter utforsking på egenhånd, har dette etter min vurdering gått veldig bra.

Jeg vil takke min veileder Anders Rønnquist for mye inspirasjon innen fagområdet, veiledning underveis og for å ha gitt mulighet for å selv være med på å styre retningen på oppgaven. En stor takk vil jeg også gi til Marcin Luczkowski som jeg har hatt gode jevnlige møter med hele semesteret. Marcin har holdt innledende kurs i relevant programvare, og kommet med gode råd underveis.

Trondheim, juni, 2018

Roger Konnestad

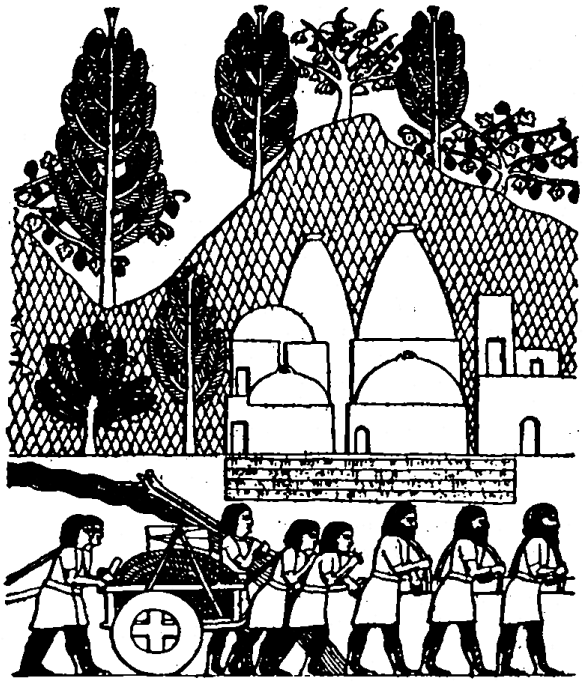
Innholdsfortegnelse

Figurliste

1 Innledning

En skallkonstruksjon kan defineres slik i følge Chris Williams: “A shell is a structure defined by a curved surface. It is thin in the direction perpendicular to the surface, but there is no absolute rule as to how thin it has to be. It might be curved in two directions, like a dome or a cooling tower, or it may be cylindrical and curve only in one direction.” [1]. Dette er en bred definisjon som inkluderer mange konstruksjonstyper som ikke er relevant for denne oppgaven. Eggeskall, skip, flykropper, brusbokser og mange flere.

Skallkonstruksjoner er en veldig gammel form for konstruksjon. Tradisjonen med å bygge kupler av murstein over runde plan går helt tilbake til neolittisk tid i Midtøsten (ca. 8 000–4 000 f.Kr. [2]). På den tiden ble de mest brukt til små boliger for de som ikke var så bemidlet og de hadde ikke noen stor rolle i monumental arkitektur. [3]



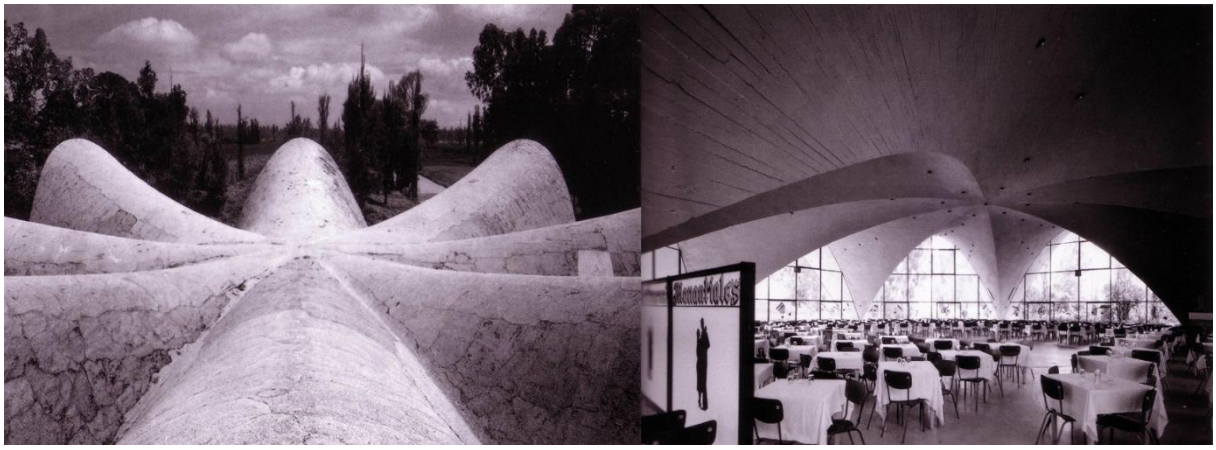
Figur 1-1: Tegning av assyrisk relief. Hentet fra: https://en.wikipedia.org/wiki/File:Vault_fig_01.gif

Romerne tok denne typen konstruksjoner til nye høyder. Med oppfinnelsen av betong og bruk av denne konstruksjonsformen, kunne man bygge mange monumentale byggverk. Pantheon er fortsatt den største uarmerte betongkuppelen i verden. [4]

St. Pauls katedral i London var et nytt gjennombrudd for denne typen konstruksjon. Robert Hooke (1635- 1703) arbeidet sammen med Christopher Wren (1632- 1723) og utviklet nye byggeteknikker. De valgte å bruke kjedelinjen snudd opp-ned for å finne den ideelle bueformen til kuppelen. Denne formen er spesielt stabil og har den egenskapen at den kun overfører trykkrefter. [1]

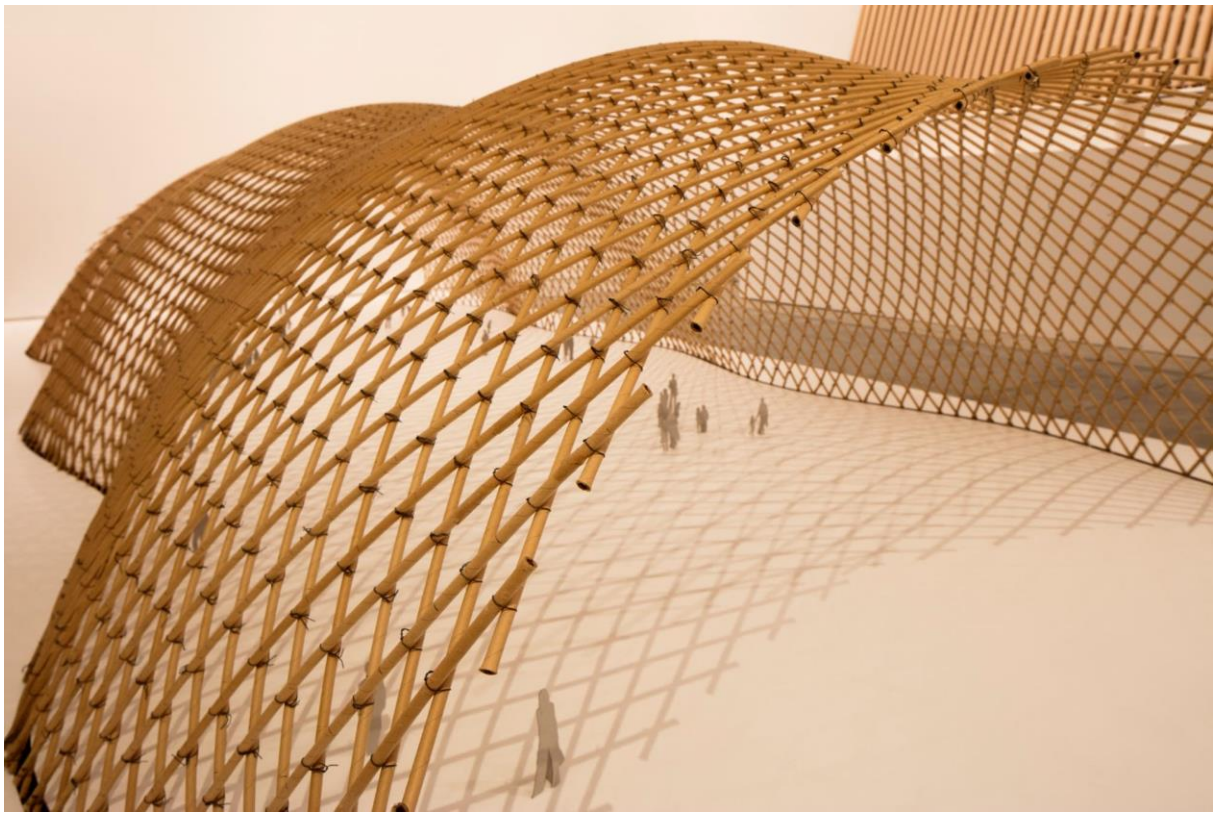
Antoni Gaudí (1852- 1926) var en pioner når det gjaldt å bruke fysiske metoder for å finne former. Når han skulle utforme Colònia Güell-krypten, kirken som aldri ble ferdig, brukte han fysiske strekk-modeller. Gaudí satte opp en modell av kirken, opp ned i 1:10 skala, ved å henge opp strenger belastet med poser med blyhagl som representerte laster. Det han da oppnådde var en strekk-konstruksjon som var optimalisert i forhold til lastene. Når han så speilet modellen i et speil som var plassert under den, ble resultatet en optimalisert trykk-konstruksjon. [5] [6]

Félix Candela (1910- 1997) var en spanskfødt arkitekt som bidro mye til utviklingen tynne skallkonstruksjoner. Han jobbet for å vise hvilket potensial tynne armerte betongskall hadde i konstruksjonsteknikk. Hans krumme hyperbolske paraboloider var stabile former som krevde minimalt med armering. Los Manantiales, som vi finner i nærheten av Mexico city, er en av hans mest kjente konstruksjoner og den har blitt kopiert flere ganger. [7]



Figur 1-2: Los Manantiales. Hentet fra: https://www.archdaily.com/496202/ad-classics-los-manantiales-felix-candela

En annen pioner innen parametrisk design er tyskeren Frei Otto (1925- 2015). Han er kjent for sine lette strekk-konstruksjoner, og var en av pionerene innen det han selv kalte "form finding". Han brukte fysiske og kjemiske sammenhenger i sine modellforsøk for å utforske ulike former. For eksempel brukte han såpefilm for å finne optimale strekkformer til flere av sine takkonstruksjoner. I samarbeid med Günter Behnisch designet han Olympiastadion i München som ble bygget for Sommer-OL i 1972. Helt til hans død i 2015 var Frei Otto en aktiv arkitekt. I årtusenskiftet på Expo 2000 i Hannover samarbeidet han med Shigeru Ban om designet av tak-strukturen til Japans paviljong, laget kun av papir. [8] [9]



Figur 1-3: Modell av Frei Otto og Shigeru Bans paviljong fra Expo 2000. Hentet fra: https://newatlas.com/shigeru-ban-emergency-shelters-scaf/48740/#gallery

Skallkonstruksjoner var mest populære i første halvdel av 1900 tallet, men har siden da falt i popularitet. Konstruksjonstypen hadde en gullalder i perioden mellom 1920 og 1960 da flere monumentale byggverk bla satt opp av arkitekter som Pier Luigi Nervi, Eduardo Torroja, Vasilii Vlassov, og Felix Candela. I tiden etter dette falt skallkonstruksjoner i popularitet. En av grunnene til dette var at kostnadene knyttet til arbeidskraft var høy. Kompliserte, store forskalinger krevde mange arbeidere med stor kompetanse. Selv med de konstruktive fordelene til skallkonstruksjoner gjorde dette at det ikke lønte seg. [10]

Nye parametriske programvarer som Grasshopper og Dynamo har revolusjonert måten konstruksjoner kan bli utformet på. Med parametrisk design på datamaskiner kan man også ta i bruk nye matematiske metoder som har blitt utviklet for å finne de mest optimale formene. Noen av disse metodene skal i denne oppgaven vurderes og sammenliknes: Force density method, Thrust network analysis og Dynamic relaxation. [1]

Skallkonstruksjoner kan lages i flere forskjellige materialer. De kan være laget av stein, betong, tre og stål. Historisk har stein og betong vært mest brukt på monumentale bygg. I senere tid har armert betong blitt mer populært. Nettverk av stål og tre har også blitt tatt mer i bruk.

Det er noen ulemper med å lage skallkonstruksjoner. Som nevnt er de dyrere og de er vanskeligere å bygge. Det er spesielt den kompliserte forskalingen som er krevende, og når det skal legges i armering blir det enda mer komplisert. Det å få et skall, med sine kurvaturer og kompliserte geometri, fra modell til virkelighet er en krevende prosess.

Fordeler med skallkonstruksjoner er at de oppleves som mer spektakulære enn konstruksjoner med et enkelt bjelke- og søylesystem. Skallkonstruksjoner er på grunn av kurvaturen mye stivere enn flate dekker. Skall kan også utformes på mer gunstige måter med tanke på lastbæring enn dekker. Colosseum kunne ikke vært bygd som et flatt tak. Det ville i tillegg til å være mindre spektakulært å se på, heller ikke vært mulig å bruke denne konstruksjonstypen med tanke på de store belastningene som taket skulle ta opp.

Figur 1-4: Byggingen av Los Manantiales av Félix Candela. Hentet fra: https://www.curbed.com/2018/1/25/16932400/felix-candela-architect-concrete-los-manantiales

2 Problemstilling

Definering av forskningsspørsmål og beskrivelse av hva oppgaven vil ta for seg.

2.1 Forskningsspørsmål

Kan bruk av nye databaserte metoder for formfinning føre til et bedre design enn tradisjonelle metoder?

2.2 Problematisering

Denne oppgaven tar for seg forskjellige metoder for å utforme og designe skallkonstruksjoner. Metodene Force density metod, Thust network analysis og Dynamic relaxation blir utforsket, testet og sammenliknet for å finne form.

Det vil bli laget parametriske modeller for å teste de forskjellige metodene med forskjellige geometrisk utgangspunkt. Målet er å finne ut hvilken metode som passer best for skallkonstruksjoner.

Oppgaven skal resultere i en oversikt og beskrivelse av ulike metoder for konstruksjon av skallkonstruksjoner. Det vil bli gitt en forklaring av utførelsen og hvordan metodene har blitt implementert i parametriske modelleringsprogrammer. Det vil deretter bli trukket en konklusjon. Denne vil ta utgangspunkt i analyseresultatene, en vurdering av fordeler og ulemper med de forskjellige metodene der man også vektlegger egenskapene til de forskjellige metodene.

3 Teori og metode

Teori og metode-kapittelet består i denne oppgaven av en teori- og en metodedel, der teorikapittelet er grunnlaget for det som gjennomføres i metodekapittelet. I teorikapittelet redegjøres det for teorien som danner grunnlaget for denne oppgaven, og i metodekapittelet diskuteres fremgangsmåten og vurderinger som er gjort underveis i prosjektet.

3.1 Teori

3.1.1 Parametrisk design

Ordet parametrisk betyr ifølge norsk grammatikk med hensyn til parametere [11]. Parametrisk design vil altså si å designe med hensyn til noen bestemte parametere. En parameter defineres av Store Norske Leksikon som en: "størrelse som kan inneha ulike verdier, men som i hvert enkelt tilfelle gis en bestemt verdi, og som påvirker utfallet av det man studerer." [12] I denne oppgaven vil det være en konstruksjon som man studerer og en parameter kan da for eksempel være belastningen på den.

3.1.2 Skall

Skall kan defineres slik som i innledningen av Chris Williams. Det er en type konstruksjon, med en tykkelse som er liten, sammenliknet med de andre dimensjonene. For å få plassert skall i sammenheng med andre typer konstruksjonsdeler kan man definere de forskjellige konstruksjonsdelene ut i fra den geometrien de har. Først tar vi for oss en rett linje. Den kan brukes til å definere for eksempel en bjelke eller en søyle. Så kan man bøye en linje slik at den blir en krum linje, som kan definere en bue. På tilsvarende vis kan et plan defineres som en plate. Man kan så bøye planet og ender da opp med et skall. Denne definisjonen vil også inkludere konstruksjoner som virker hovedsakelig i strekk, strekkonstruksjoner. Derfor lages det ofte en egen kategori for konstruksjoner som virker i strekk som inkluderer telt, seil og ballonger. [1]

Det kan skilles mellom aktive former og passive former. En aktiv form forandrer sin form når den blir utsatt for belastninger. Dette i motsetning til en passiv form som ikke forandrer sin form i vesentlig grad under belastning. En skallkonstruksjon som virker i trykk, og som har en passiv form, vil være det denne oppgaven vil ta for seg. [1]

Skallkonstruksjoner kan enten være kontinuerlige eller gitterskall. Los Manantiales av Félix Candela, som er nevnt i innledningen, er en typisk kontinuerlig skallkonstruksjon. Den er laget av betong og er helt kontinuerlig slik at kreftene fordeler seg fritt i konstruksjonen. Et gitterskall kan for eksempel se ut slik som i British Museum i London på bilde under. Et gitterskall har de

samme egenskapene som et kontinuerlig skall med tanke på geometri og strukturell oppførsel, men i et gitterskall er det flere «hull» i konstruksjonen som begrenser hvordan kreftene har mulighet for å bevege seg. [1]

Figur 3-1: British Museum, London. Hentet fra: https://londonist.com/london/secret/facts-about-the-british-museum

3.1.3 Fysiske metoder for å finne form

Lenge før de moderne teoriene innen konstruksjon oppsto brukte antikkens arkitekter ulike metoder basert på proporsjoner og geometri for å finne formen til sine bygninger. I Romerriket lagde man sylindriske og halvkuleformede bygninger, og i middelalderen begynte man å konstruere spissbuer. Disse designene ble beskrevet av enkle geometriske former, som sirkler, firkanter og trekanter. Enkle former ble benyttet for å finne form og dimensjoner på det som skulle bygges. Lenge trodde man at bygningene var så effektive fordi man brukte disse enkle formene, som var de samme formene som Gud hadde brukt for å lage verden. [1]

Dersom man holder en kjede fast i hver ende danner kjeden en naturlig kurve på grunn av tyngdekraften som kalles kjedelinje eller katenær kurve. [13] Denne kjeden vil da bare virke i strekk. Dersom man snur denne kurven opp ned får man en bue som virker bare i trykk. Dette oppdaget Robert Hooke i 1676. [1]

Ved å bruke slike fysiske metoder finner man ut hvilke former konstruksjonene må ha for å være i likevekt. Det vil si at de ytre og indre kreftene er like slik at man får en stabil konstruksjon som ikke forandrer form i stor grad under belasting.

3.1.4 Digitale metoder for å finne form

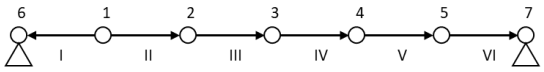
På 60 og 70 tallet begynte det å dukke opp matematiske metoder for å finne likevekt i en konstruksjon. Det er tre av disse metodene som skal undersøkes og sammenliknes i denne oppgaven: Force density method, Thrust network analysis og Dynamic relaxation. Disse metodene fungerer i prinsippet på samme måte som de fysiske metodene. Man bestemmer opplagerbetingelser, laster og hva slags form man skal ta utgangspunkt i og så kommer det en ny geometri ut som er i likevekt. Det finnes flere slike metoder enn de som blir beskrevet i denne oppgaven og det lages stadig nye versjoner av metodene.

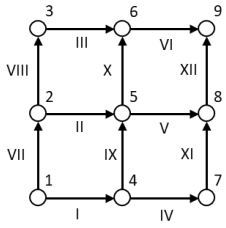
3.1.5 Gren-node matriser

Alle metodene som skal undersøkes i denne oppgaven vil ta i bruk det som kalles for gren-node matriser. Dette er matriser som beskriver forholdet mellom staver og noder i konstruksjonen man skal finne likevekt i. Gren-node matrisen, C_{ij}, settes opp for å holde orden på hvordan strukturen er satt sammen. Hver node nummereres og i matrisen beskrives det i hvilken node hver enkelt linje starter og slutter på denne måten:

$$C_{ij} = \begin{cases} +1 \; hvis \; gren \; j \; ender \; i \; node \; i \\ -1 \; hvis \; gren \; j \; begynner \; i \; node \; i \\ 0 \; ellers \end{cases}$$

Under vises det to eksempler på hvordan gren-node matriser kan settes opp ut i fra en gitt geometri:





$$\boldsymbol{C} = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} \text{I} \\ \text{II} \\ \text{III} \\ \text{IV} \\ \text{V} \\ \text{VI} \end{matrix}$$

$$\boldsymbol{C} = \begin{array}{c} \begin{matrix} 2 & 4 & 5 & 6 & 8 & 1 & 3 & 7 & 9 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} \text{I} \\ \text{II} \\ \text{III} \\ \text{IV} \\ \text{V} \\ \text{VI} \\ \text{VII} \\ \text{VIII} \\ \text{IX} \\ \text{X} \\ \text{XI} \\ \text{XII} \end{matrix}$$

3.1.6 Notasjon

I utledningen av metodene som skal utledes brukes mye av den samme notasjonen for å beskrive matrisene. Det skilles mellom stor og liten bokstav der liten a betyr at den er diagonalen av stor A:

$$\boldsymbol{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = [1 \quad 2 \quad 3]^T, \boldsymbol{a}^T = [1 \quad 2 \quad 3]$$

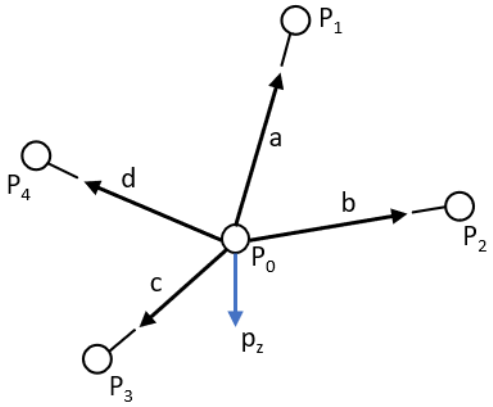
$$\boldsymbol{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \boldsymbol{A} = diag(\boldsymbol{a}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

3.1.7 Force density method (FDM)

Force density method er en metode som brukes for å beregne likevekt i en konstruksjon. Den ble utviklet som en respons på behovet for datamodellering av bygninger til Olympiaparken i München [14]. FDM bygger på arbeidet til Avinadav Siev fra 1963 [15] og ble først introdusert av Klaus Linkwitz og Hans-Jörg Schek i 1972 [16]. Metoden bygger på antagelsen om at forholdet mellom strekkraften og lengden av hver kabel er konstant, som gjør et system av ikke-lineære ligninger om til et sett med lineære ligninger som kan løses direkte. FDM er en metode som ble utviklet for kabelnett og brukes mye i strekkonstruksjoner, men den kan også brukes på skallkonstruksjoner.

Videre følger en utledning og formulering av metoden slik den er forklart av Klaus Linkwitz i «Shell Structures for Architecture» [1].

Formulering av likevekt



Figur 3-2: Node med krefter og last. Figur laget i PowerPoint.

Hookes lov:

$$F_i = \left[\frac{EA}{l_0} \cdot e\right], i = a, b, c, d$$

Lengden til hver stav bestemmes ut i fra nodene:

$$l_i = \sqrt{(x_k - x_0)^2 + (y_k - y_0)^2 + (z_k - z_0)^2}, k = 1, 2, 3, 4$$

Elastisk forlengelse:

$$e_i = l_i - l_{0,i}$$

Likevekt i nodene (gjøres også i y- og z-retning):

$$F_{i,x} = F_i \cdot \cos \alpha_i$$

$\cos \alpha_i$ er retningscosinus og α_i er vinkelen mellom stav i og x-retningen. Likevekt av de fire kreftene i P_0 i x-retning:

$$F_a \cdot \cos \alpha_a + \ldots + F_d \cdot \cos \alpha_a + p_x = 0$$

$$direction\ \cos(retningscosinus) = \frac{coordinate\ difference(koordinat\ forskjell)}{distance\ in\ space(avstand)}$$

Bytter ut retningscosinus:

$$\frac{x_1 - x_0}{l_a} \cdot F_a + \ldots + \frac{x_4 - x_0}{l_d} \cdot F_d + p_x = 0$$

Elastisk forlengelse i Hookes lov:

$$F_i = \left[\frac{EA}{l_0} \cdot (l - l_0)\right]$$

Hookes lov i likevekten:

$$\frac{x_1 - x_0}{l_a} \cdot \frac{EA_a}{l_{0,a}} \cdot (l_a - l_{0,a}) + \ldots + \frac{x_4 - x_0}{l_d} \cdot \frac{EA_d}{l_{0,d}} \cdot (l_d - l_{0,d}) + p_x = 0$$

Krafttetthet defineres for å løse det ikke-lineære problemet:

$$force\ density = \frac{force\ in\ a\ bar}{stressed\ length\ of\ the bar}$$

Krafttetthet i likevekten:

$$(x_1 - x_0) \cdot \frac{F_a}{l_a} + \ldots + (x_4 - x_0) \cdot \frac{F_d}{l_d} + p_x = 0$$

$$q_i \coloneqq \frac{F_i}{l_i}$$

$$(x_1 - x_0) \cdot q_a + \ldots + (x_4 - x_0) \cdot q_d + p_x = 0$$

Ukjente verdier samles på venstre side:

$$-(q_a + q_b + q_c + q_d) \cdot x_0 = -p_x - (x_1 \cdot q_a + x_2 \cdot q_b + x_3 \cdot q_c + x_4 \cdot q_d)$$

Med løsningen i x-, y- og z-retning:

$$x_0 = \frac{p_x + x_1 \cdot q_a + x_2 \cdot q_b + x_3 \cdot q_c + x_4 \cdot q_d}{q_a + q_b + q_c + q_d}$$

$$y_0 = \frac{p_y + y_1 \cdot q_a + y_2 \cdot q_b + y_3 \cdot q_c + y_4 \cdot q_d}{q_a + q_b + q_c + q_d}$$

$$z_0 = \frac{p_z + z_1 \cdot q_a + z_2 \cdot q_b + z_3 \cdot q_c + z_4 \cdot q_d}{q_a + q_b + q_c + q_d}$$

For noden P_0 og dens koblinger blir C matrisen:

$$\boldsymbol{C} = \begin{bmatrix} +1 & -1 & 0 & 0 & 0 \\ +1 & 0 & -1 & 0 & 0 \\ +1 & 0 & 0 & -1 & 0 \\ +1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

Koordinat forskjell regnes ut slik:

$$\boldsymbol{u} = \boldsymbol{C}\boldsymbol{x}$$

$\boldsymbol{C}$ deles opp i $\boldsymbol{C}_N$ og $\boldsymbol{C}_F$ for å skille mellom nye punker, og faste punkter. Tilsvarende for $\boldsymbol{x}$:

$$\boldsymbol{C} = [\boldsymbol{C}_N \quad \boldsymbol{C}_F]$$

$$\boldsymbol{x} = [\boldsymbol{x}_N \quad \boldsymbol{x}_F]$$

Koordinat forskjellene blir da:

$$\boldsymbol{u} = \boldsymbol{C}_N\boldsymbol{x}_N + \boldsymbol{C}_F\boldsymbol{x}_F$$

$$\boldsymbol{u} = \boldsymbol{C}_N\boldsymbol{y}_N + \boldsymbol{C}_F\boldsymbol{y}_F$$

$$\boldsymbol{u} = \boldsymbol{C}_N\boldsymbol{z}_N + \boldsymbol{C}_F\boldsymbol{z}_F$$

Som blir brukt til å finne stavlengdene:

$$\boldsymbol{L} = (\boldsymbol{U}^2 + \boldsymbol{V}^2 + \boldsymbol{W}^2)^{\frac{1}{2}}$$

For å få likningen fra likevekten tidligere på matrise-form må Jakobideterminanten regnes ut:

$$\boldsymbol{f}(x_0) = \begin{bmatrix} f_a(x_0) \\ f_b(x_0) \\ f_c(x_0) \\ f_d(x_0) \end{bmatrix} = \begin{bmatrix} l_a \\ l_b \\ l_c \\ l_d \end{bmatrix} = \boldsymbol{l}$$

$$\left(\frac{\partial \boldsymbol{f}(x_0)}{\partial x_0}\right)^T = \begin{bmatrix} \frac{\partial l_a}{\partial x_0} & \frac{\partial l_b}{\partial x_0} & \frac{\partial l_{ac}}{\partial x_0} & \frac{\partial l_d}{\partial x_0} \end{bmatrix} = \begin{bmatrix} \frac{-(x_1 - x_0)}{l_a} \\ \frac{-(x_2 - x_0)}{l_b} \\ \frac{-(x_3 - x_0)}{l_c} \\ \frac{-(x_4 - x_0)}{l_d} \end{bmatrix}^T$$

$$\left(\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)^T = \boldsymbol{C}_N^T\boldsymbol{U}\boldsymbol{L}^{-1}$$

Likevekten på matrise-form:

$$-\left(\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)^{T}\begin{bmatrix}F_a\\F_b\\F_c\\F_d\end{bmatrix}+p_x=0$$

Omskrevet:

$$\boldsymbol{C_N^T U L^{-1} f + p = 0}$$

Krafttetthet på matriseform og satt inn i likningen over:

$$\boldsymbol{q = L^{-1} f}$$

$$\boldsymbol{C_N^T U q + p = 0}$$

Gitt at:

$$\boldsymbol{Uq = Qu = QCx}$$

Blir det da:

$$\boldsymbol{C_N^T Q C x + p = 0}$$

Med oppdelte matriser:

$$\boldsymbol{C_N^T Q C_N x_N + C_N^T Q C_F x_F + p = 0}$$

For å forenkle:

$$\boldsymbol{D_N = C_N^T Q C_N}$$

$$\boldsymbol{D_F = C_N^T Q C_F}$$

Med forenklingene blir det:

$$\boldsymbol{D_N x_N = p - D_F x_F}$$

Ved å stokke om får man likningen for de nye koordinatene:

$$\boldsymbol{x_N = D_N^{-1}(p_x - D_F x_F)}$$

$$\boldsymbol{y_N = D_N^{-1}(p_y - D_F y_F)}$$

$$\boldsymbol{z_N = D_N^{-1}(p_z - D_F z_F)}$$

3.1.8 Thrust network analysis (TNA)

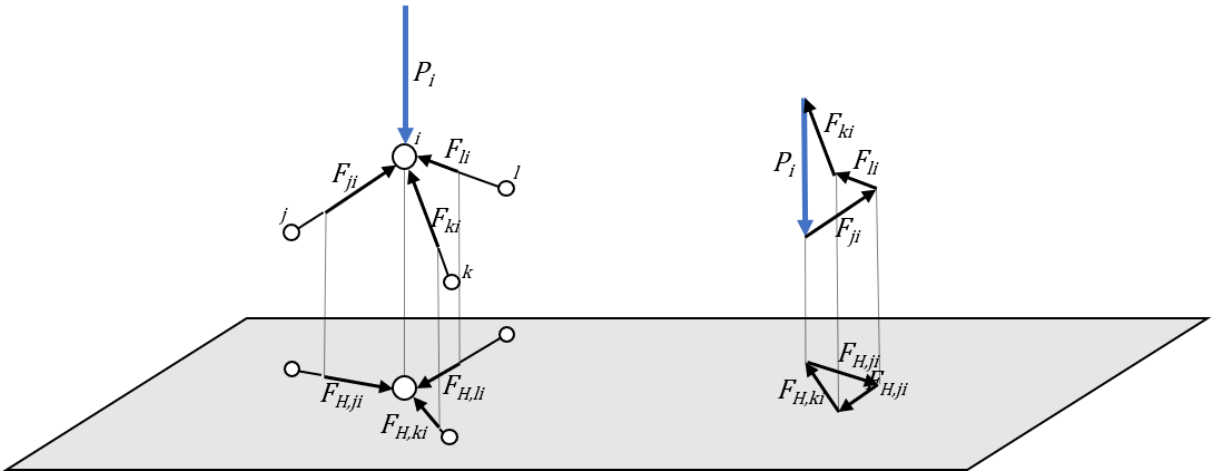
Thrust network analysis er en metode som ble introdusert av Philippe Block i 2009. [17] Den bygger på grafisk statikk, som er en måte å finne likevekt på ved hjelp av form og kraft diagrammer. Denne metoden kan brukes til å finne trykklinjer. Trykklinjer kan brukes til å visualisere hvordan trykkreftene beveger seg i en konstruksjon, hvis linjen ikke beveger seg på utsiden av konstruksjonen er det en form som kan fungere. [1]

Thrust network analysis gjør det mulig å utvide grafisk statikk til 3D. Den utvider konseptet til et nettverk av trykklinjer med tyngdekraft som belastning. Metoden finner dermed ut om det finnes en mulig likevekt for konstruksjonen. Knekking, bøyning, asymmetriske laster må dermed sjekkes i en separat analyse etterpå. Siden TNA finner trykklinjer passer den bra til å finne former til skall som bare skal virke i trykk, spesielt uarmerte hvelvede konstruksjoner.

Videre følger en utledning og formulering av metoden slik den er forklart av Philippe Block, Lorenz Lachauer og Matthias Rippmanni i «Shell Structures for Architecture» [1].

Formulering av likevekt

Likevekt av en node i i $\boldsymbol{G}$ (thrust network)



Figur 3-3: Node i med formdiagram (til venstre) og kraftdiagram (til høyre). Figur laget i PowerPoint.

$$F_{H,ji} + F_{H,ki} + F_{H,li} = 0$$

$$F_{V,ji} + F_{V,ki} + F_{V,li} = P_i$$

Siden det er en sammenheng mellom form- og kraftdiagrammet kan de horisontale komponentene skrives som en funksjon av de korresponderende lengdene og en skaleringsfaktor:

$$F_{H,ji} = \frac{1}{r} \cdot l^*_{H,ji}$$

Grenlengdene:

$$l^*_{H,ji} = \sqrt{\left(x^*_i - x^*_j\right)^2 + \left(y^*_i - y^*_j\right)^2}$$

Den vertikale likevekten kan skrives om til en funksjon av $F_{H,ji}$ og geometrien til $\boldsymbol{G}$:

$$F_{H,ji} \cdot \frac{z_i - z_j}{I_{H,ji}} + F_{H,ki} \cdot \frac{z_i - z_k}{I_{H,ki}} + F_{H,li} \cdot \frac{z_i - z_l}{I_{H,li}} = P_i$$

$$l_{H,ji} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}$$

Likevekten med de horisontale komponentene byttet ut og omskrevet:

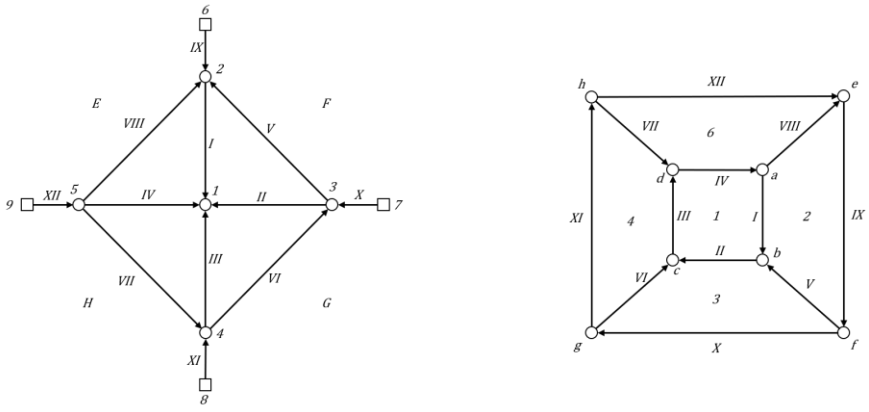
$$l^*_{H,ji} \cdot \frac{z_i - z_j}{I_{H,ji}} + l^*_{H,ki} \cdot \frac{z_i - z_k}{I_{H,ki}} + l^*_{H,ji} \cdot \frac{z_i - z_l}{I_{H,li}} = P_i \cdot r$$

$$\left(\frac{I^*_{H,ji}}{I_{H,ji}} + \frac{I^*_{H,ki}}{I_{H,ki}} + \frac{I^*_{H,li}}{I_{H,li}}\right) \cdot z_i - \frac{I^*_{H,ji}}{I_{H,ji}} \cdot z_j - \frac{I^*_{H,ki}}{I_{H,ki}} \cdot z_k - \frac{I^*_{H,li}}{I_{H,li}} \cdot z_l - P_i \cdot r = 0$$

Substituerer:

$$d_i \cdot z_i - d_j \cdot z_j - d_k \cdot z_k - d_l \cdot z_l - P_i \cdot r = 0$$

<u>Formulering av matriser</u>



Figur 3-4: Formdiagram (til venstre) og kraftdiagram (til høyre). Figur laget i PowerPoint.

Form- og kraftmatriser:

$$\boldsymbol{C} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$$\boldsymbol{C}^* = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

Likevekten på matriseform:

$$\boldsymbol{C}_N^T(\boldsymbol{L}_H^{-1}\boldsymbol{L}_H^*)\boldsymbol{C}\boldsymbol{z} - r\boldsymbol{p} = \boldsymbol{C}_N^T(\boldsymbol{T})\boldsymbol{C}\boldsymbol{z} - r\boldsymbol{p} = \boldsymbol{0}$$

Sammenhengen mellom krafttetthet $\boldsymbol{q}$ og parameteren $\boldsymbol{t}$:

$$\boldsymbol{q} = \frac{1}{r} \cdot \boldsymbol{L}_H^{-1}\boldsymbol{l}_H^* = \frac{1}{r} \cdot \boldsymbol{t}$$

Definering av $\boldsymbol{D}$:

$$\boldsymbol{D} = \boldsymbol{C}_N^T\boldsymbol{T}\boldsymbol{C}$$

Med $\boldsymbol{D}$ kan likevekten omskrives:

$$\boldsymbol{D}\boldsymbol{z} - r\boldsymbol{p} = \boldsymbol{D}_N\boldsymbol{z}_N - \boldsymbol{D}_F\boldsymbol{z}_F - r\boldsymbol{p} = \boldsymbol{0}$$

$$\boldsymbol{z}_N = \boldsymbol{D}_N^{-1}(\boldsymbol{p}r - \boldsymbol{D}_F\boldsymbol{z}_F)$$

3.1.9 Dynamic relaxation (DR)

Dynamic relaxation ble oppfunnet av Alistair Day i 1965. [1] Det er en numerisk metode som ender opp med å løse et sett med ikke-lineære likninger. Metoden følger bevegelsen til en konstruksjon over tid, under belastning, for å finne likevekt. Man får en svingning i konstruksjonen som beskrives ved hjelp av fart og akselerasjon og demping kan legges til i metoden for å gjøre den mer effektiv med hensyn på bruk av datakraft. [18] I DR brukes variabler som er intuitive for en ingeniør. Verdier som aksialstivhet (EA), bøyestivhet (EI), brukes direkte og på grunnlag av dette kan man gjøre statiske analyser direkte.

Videre følger en utledning og formulering av metoden slik den er forklart av Sigrid Adriaenssens, Mike Barnes, Richard Harris og Chris Williams i «Shell Structures for Architecture» [1].

Formulering

Newtons 2. lov:

$$R_{ix}^{t} = M_i \dot{v}_{ix}^{t}$$

Hvor massen M_i, som er samlet i nodene, er:

$$M_i = \frac{\Delta t^2}{2} S_i$$

Stivheten S_i er:

$$S_i = \sum_{m=1}^{k} \left(\frac{EA^S}{L_0} + G \frac{T^S}{L^S} \right)$$

Omskriving av Newtons 2. lov med fart i stedet for akselerasjon:

$$v_{ix}^{t+\Delta t/2} = v_{ix}^{t-\Delta t/2} + \frac{\Delta t}{M_i} R_{ix}^{t}$$

Den oppdaterte geometrien blir da:

$$x_i^{t+\Delta t} = x_i^{t} + \Delta t \cdot v_{ix}^{t+\Delta t/2}$$

Kreftene kan finnes ved:

$$R_{ix}^{t+\Delta t} = P_{ix} + \sum_{m=1}^{k} \left[\left(\frac{F}{L} \right) (x_j - x_i) \right]^{t+\Delta t}$$

Med viskøs demping blir farten:

$$v_{ix}^{t+\Delta t/2} = A \cdot v_{ix}^{t-\frac{\Delta t}{2}} + B \cdot \frac{\Delta t}{M_i} R_{ix}^{t}$$

Der C er en konstant. I tilfeller med bare kinetisk demping er $A = 1$.

$$A = (1 - C/2)/(1 + C/2)$$

$$B = (1 + A)/2$$

3.2 Metode

3.2.1 Programvare

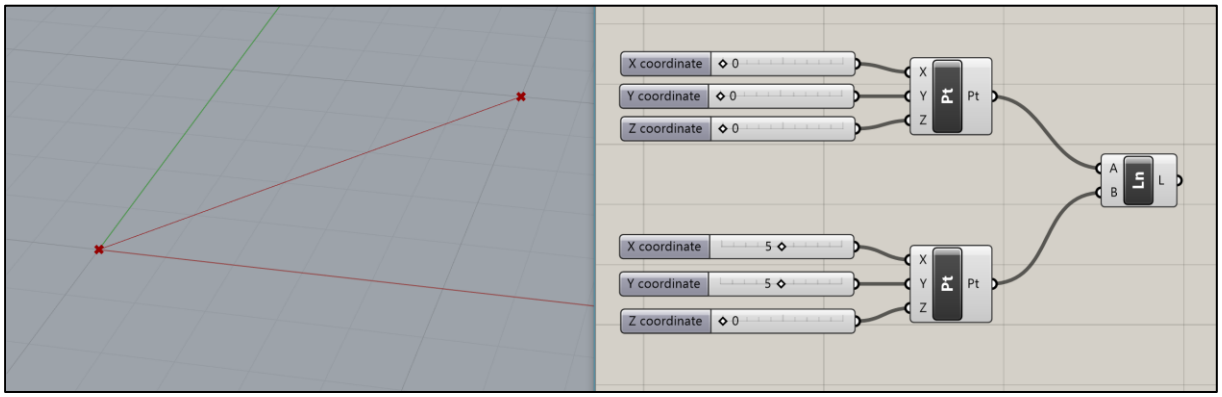
Får å få testet og sammenliknet metodene ble det brukt en kombinasjon av flere programvarer. Her følger en oversikt over relevant programvare brukt i prosjektet.

Rhinoceros 3D

Rhinoceros er en kommersiell 3D-datagrafikk og datamaskinstøttet design programvare. Rhino 3D ble utviklet av Robert McNeel & Associates, et amerikansk selskap grunnlagt i 1980. Programmet ble brukt til å lage modeller i 2D og 3D, og geometri som ble funnet ved hjelp av andre tillegg ble visualisert i Rhino.

Grasshopper

Grasshopper er et visuelt programmeringsspråk som er utviklet av David Rutten ved Robert McNeel & Associates. Det er en grafisk algoritme som er tett integrert med Rhino's 3-D modelleringsverktøy. Programmet virker ved å dra komponenter og koblinger mellom dem på et lerret. Utgangene til disse komponentene blir da koblet til inngangene til etterfølgende komponenter og man kan på denne måten lage geometri som vil visualiseres i Rhino. På bildet under illustreres et eksempel der en linje blir laget ved hjelp av to punkter som igjen er definert av verdiene til koordinatene. Når man modellerer på denne måten kan man gjøre modellen parametrisk.



Figur 3-5: En linje blir definert i Grasshopper i vinduet til høyre og geometrien vises i Rhino i vinduet til venstre.

Karamba

Karamba er et tillegg til Grasshopper for parametriske konstruksjonsanalyser. Karamba ble utviklet av Clemens Preisinger og kan analysere både fagverk, rammer og skall. Når en konstruksjon er laget i Grasshopper kan modellen analyseres direkte i programmet, uten at den må eksporteres til et annet analyseprogram. Karamba består av flere komponenter som virker i Grasshopper, og når man har modellen i Grasshopper kan man bare dra en linje fra utgangen til geometrien over til en Karamba-komponent. Der kan man legge til opplagere, last, tverrsnitt og materiale. Så tas dette videre i analyser som gir data for nedbøyning, krefter og momenter.

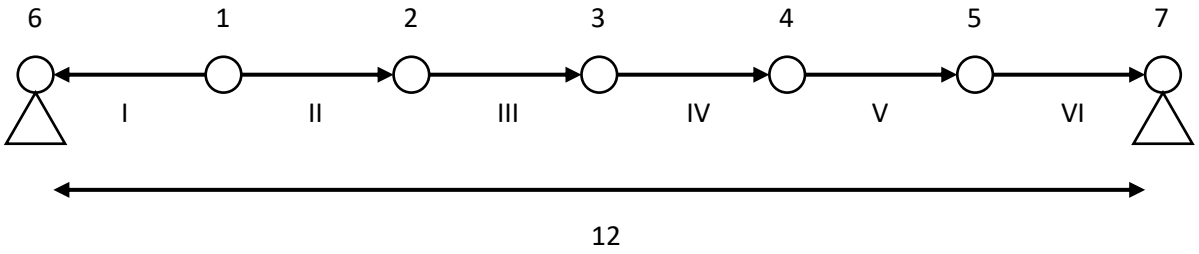
Microsoft Visual Studio

Visual Studio er et integrert utviklingsmiljø for Microsofts .NET platform. Programmet hjelper utviklere å utvikle programmer for Windows, såkalte Windows Forms, websider og mobil programvare for Microsofts mobile operativsystemer. Visual Studio støtter følgende programmeringsspråk: Visual Basic, Visual C++, Visual C# og Visual J#.

I denne oppgaven har Visual Studio blitt brukt til å programmere i C#. Ved hjelp av Visual studio og C# kan man lage komponenter som kan brukes i Grasshopper. På denne måten kunne formfinning metodene programmeres og legges til som komponenter i Grasshopper. For å kunne gjøre nødvendige matematiske beregninger ble Math.NET, som er et tillegg i Visual Studio, tatt i bruk.

3.2.2 Eksempel 1: Formfinning i 2D

For å sammenlikne metodene ble det laget en komponent av hver type i C# for Grasshopper. De første komponentene som ble laget var for 2D-bjelker. Ved å lage dem i 2D kunne komponentene sammenliknes på et grunnleggende nivå uten at det var for mye komplisert geometri å ta hensyn til. Vi tar i dette eksempelet utgangspunkt i en bjelke på 12 enheters lengde bestående av 7 noder. Under vises utregning for et slikt tilfelle der vi bruker Force density method til å finne ny optimalisert geometri.



Figur 3-6: Geometrien til en 2D bjelke. Figuren er laget i PowerPoint.

Definering $\boldsymbol{C}$-matrisen og koordinat vektorene $\boldsymbol{x}$ og $\boldsymbol{y}$

$$\boldsymbol{C} = [\boldsymbol{C}_N \quad \boldsymbol{C}_F] = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{x} = [\boldsymbol{x}_N \quad \boldsymbol{x}_F] = [-4 \quad -2 \quad 0 \quad 2 \quad 4 \quad -6 \quad 6]^T$$

$$\boldsymbol{y} = [\boldsymbol{y}_N \quad \boldsymbol{y}_F] = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$$

Koordinat forskjellene satt opp som $\boldsymbol{u}$ og $\boldsymbol{v}$

$$\boldsymbol{u} = \boldsymbol{Cx}, \boldsymbol{v} = \boldsymbol{Cy}$$

$$\boldsymbol{u} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} -4 \\ -2 \\ 0 \\ 2 \\ 3 \\ -6 \\ 6 \end{bmatrix} = [2 \quad -2 \quad -2 \quad -2 \quad -2 \quad -2]^T$$

$$\boldsymbol{v} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$$

$\boldsymbol{u}$ og $\boldsymbol{v}$ er diagonalene av $\boldsymbol{U}$ og $\boldsymbol{V}$ som brukes til å finne grenlengden $\boldsymbol{L}$ som har diagonalen $\boldsymbol{l}$

$$\boldsymbol{L} = (\boldsymbol{U}^2 + \boldsymbol{V}^2)^{\frac{1}{2}}$$

$$\boldsymbol{l} = [2 \quad -2 \quad -2 \quad -2 \quad -2 \quad -2]^T$$

Nå er det klart for å ta i bruk Force density method

Likevekt mellom lastene $\boldsymbol{p}_x$ og $\boldsymbol{p}_y$ og de indre kreftene $\boldsymbol{f}_x$ og $\boldsymbol{f}_y$

$$\boldsymbol{p}_x - \boldsymbol{f}_x = 0$$

$$\boldsymbol{p}_y - \boldsymbol{f}_y = 0$$

De indre kreftene regnes ut fra gren kreftene f (eksempel)

$$p_{x,1} = f_{x,1} = f_{x,I} + f_{x,II} = f_I \frac{u_I}{l_I} + f_{II} \frac{u_{II}}{l_{II}}$$

$$p_{y,1} = f_{y,1} = f_{y,I} + f_{y,II} = f_I \frac{v_I}{l_I} + f_{II} \frac{v_{II}}{l_{II}}$$

For hele nettverket

$$\boldsymbol{p}_x - \boldsymbol{f}_x = \boldsymbol{p}_x - \boldsymbol{C}_N^T \boldsymbol{U} \boldsymbol{L}^{-1} \boldsymbol{f} = \boldsymbol{0}$$

$$\boldsymbol{p}_y - \boldsymbol{f}_y = \boldsymbol{p}_y - \boldsymbol{C}_N^T \boldsymbol{V} \boldsymbol{L}^{-1} \boldsymbol{f} = \boldsymbol{0}$$

Ytre last defineres

$$\boldsymbol{p}_x = 0$$

$$\boldsymbol{p}_y = [1 \quad 1 \quad 1 \quad 1 \quad 1]^T$$

Krafttettheten defineres

$$\boldsymbol{q} = \boldsymbol{L}^{-1} \boldsymbol{f} = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]^T$$

Omskriving av likevekten der $\boldsymbol{Q}$ er den diagonale matrisen satt sammen av $\boldsymbol{q}$

$$\boldsymbol{C}_N^T \boldsymbol{U} \boldsymbol{q} = \boldsymbol{C}_N^T \boldsymbol{Q} \boldsymbol{C} \boldsymbol{x} = \boldsymbol{C}_N^T \boldsymbol{Q} \boldsymbol{C}_N \boldsymbol{x}_N + \boldsymbol{C}_N^T \boldsymbol{Q} \boldsymbol{C}_F \boldsymbol{x}_F = \boldsymbol{p}_x$$

$$\boldsymbol{C}_N^T \boldsymbol{V} \boldsymbol{q} = \boldsymbol{C}_N^T \boldsymbol{Q} \boldsymbol{C} \boldsymbol{y} = \boldsymbol{C}_N^T \boldsymbol{Q} \boldsymbol{C}_N \boldsymbol{y}_N + \boldsymbol{C}_N^T \boldsymbol{Q} \boldsymbol{C}_F \boldsymbol{y}_F = \boldsymbol{p}_y$$

For enkelhets skyld regnes $\boldsymbol{D}_N$, $\boldsymbol{D}_F$ og $\boldsymbol{D}_N^{-1}$ut først

$$\boldsymbol{D}_N = \boldsymbol{C}_N^T \boldsymbol{Q} \boldsymbol{C}_N = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

$$\boldsymbol{D}_F = \boldsymbol{C}_N^T \boldsymbol{Q} \boldsymbol{C}_F = \begin{bmatrix} -1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\boldsymbol{D}_N^{-1} = \begin{bmatrix} \frac{5}{6} & \frac{2}{3} & \frac{1}{2} & \frac{1}{3} & \frac{1}{6} \\ \frac{2}{3} & \frac{4}{3} & 1 & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{2} & 1 & \frac{3}{2} & 1 & \frac{1}{2} \\ \frac{1}{3} & \frac{2}{3} & 1 & \frac{4}{3} & \frac{2}{3} \\ \frac{1}{6} & \frac{1}{3} & \frac{1}{2} & \frac{2}{3} & \frac{5}{6} \end{bmatrix}$$

Med dette er alt klart for å regne ut de nye koordinatene $\boldsymbol{x}_N$ og $\boldsymbol{y}_N$

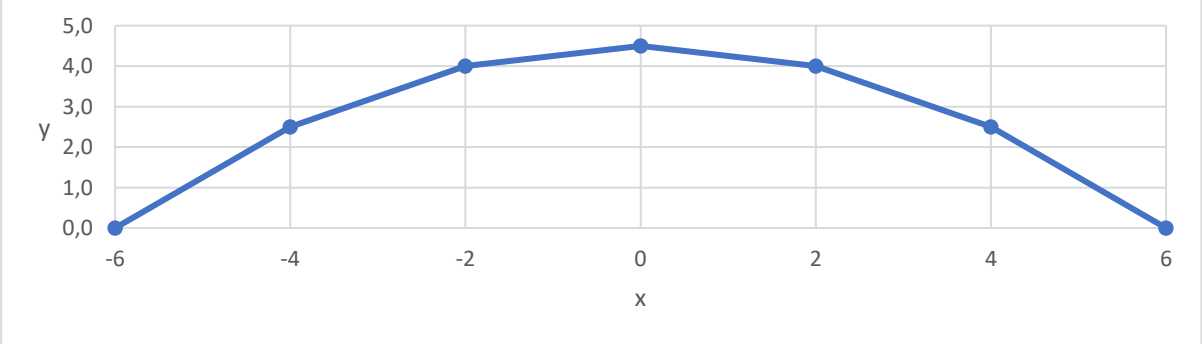
$$\boldsymbol{x}_N = \boldsymbol{D}_N^{-1}(\boldsymbol{p}_x - \boldsymbol{D}_F \boldsymbol{x}_F)$$

$$\boldsymbol{y}_N = \boldsymbol{D}_N^{-1}(\boldsymbol{p}_y - \boldsymbol{D}_F \boldsymbol{y}_F)$$

$$\boldsymbol{x}_N = [-4 \quad -2 \quad 0 \quad 2 \quad 4]^T$$

$$\boldsymbol{y}_N = [2.5 \quad 4 \quad 4.5 \quad 4 \quad 2.5]^T$$

Resultatet plottet



Figur 3-7: Resultatet plottet i Excel.

3.2.3 Force density method i 2D

Den første komponenten som ble laget var Force density method for 2D eksempelet beskrevet i 3.2.2. Det første som måtte gjøres var å bestemme inndata til komponenten. Siden utgangspunktet skulle være en rett linje med to opplagere var all inndata som var nødvendig for geometrien lengden på linjen og antall segmenter. De to siste inndataene var for den ytre kraften som skulle virke i negativ z-retning og krafttettheten. Dette ble da parameterne som kunne variere.

Det første som måtte settes opp var vektorene for koordinatene. For x-koordinatene ble det gjort ved først å lage en liste med verdier som ble laget av en løkke. Løkken gikk fra minus linjelengden delt på to til pluss linjelengden delt på to. Løkken økte med linjelengden delt på antall segmenter for å treffe i alle nodene.

```
List<double> Xnl = new List<double>();
for (double i = (-dis / 2); i <= (dis / 2); i += (dis / seg))
{
    Xnl.Add(i);
}


List<double> Xfl = new List<double>();
Xfl.Add(Xnl[0]);
Xfl.Add(Xnl[Xnl.Count - 1]);


Xnl.RemoveAt(0);
Xnl.RemoveAt(Xnl.Count - 2);
```

Kode 3-1: Listene med verdiene til x-koordinatene blir satt opp.

For å sette opp vektorene ble Math.NET tillegget tatt i bruk.

```
var M = Matrix<double>.Build;
var xn = M.DenseOfColumnMajor(Xnl.Count, 1, Xnl.ToArray());
var xf = M.DenseOfColumnMajor(Xfl.Count, 1, Xfl.ToArray());
```

*Kode 3-2: Vektorene **Xn** og **Xf** blir satt opp.*

For y-koordinatene var det bare å sette opp vektoren direkte, siden utgangspunktet var en flat linje er alle y-koordinatene lik 0.

```
var yn = M.Dense(Xnl.Count, 1, 0);
var yf = M.Dense(Xfl.Count, 1, 0);
```

*Kode 3-3: Vektorene **Yn** og **Yf** blir satt opp.*

En av de største utfordringene med å programmere FDM er å sette opp C-matrisen (gren-node matrisen). Når inndataen bare er en linje med punkter er det mulig å se et mønster i hvordan C-matrisen blir. Det vil alltid være to diagonale linjer med -1 og +1 i et fast mønster avhengig av antall punkter på linjen. For å lage C-matrisen ble det laget en liste med alle verdiene som skulle være i matrisen og disse ble så satt opp i en matrise.

```
List<double> d = new List<double>();
d.Add(1);
d.Add(1);
for (int i = 0; i < (Xnl.Count - 1); i++)
{
    for (int j = 0; j < Xnl.Count; j++)
    {
    d.Add(0);
    }
    d.Add(-1);
    d.Add(1);
}
```

Kode 3-4: Listen med verdiene som skal brukes i C_n-matrisen settes opp.

```
List<double> g = new List<double>();
g.Add(-1);
for (int j = 0; j < (Xnl.Count * 2); j++)
{
    g.Add(0);
}
g.Add(-1);
```

Kode 3-5: Listen med verdiene som skal brukes i C_f-matrisen settes opp.

```
var Cn = M.DenseOfColumnMajor((Xnl.Count + 1), Xnl.Count, d.ToArray());
var Cf = M.DenseOfColumnMajor((Xnl.Count + 1), Xfl.Count, g.ToArray());
```

Kode 3-6: Matrisene blir satt opp fra listene som er laget.

De neste matrisene som måtte settes opp var for lastene, en for x-retning, p_x, og en for y-retning, p_y. I x-retning vil det i dette eksemplet ikke være noen krefter, så for den matrisen ble alle verdiene lik 0. For y-retning ble det laget en egen inndataliste med verdier for lasten i hvert enkelt punkt.

```
var Px = M.Dense(Xnl.Count, 1, 0);
var Py = M.Dense(Xnl.Count, 1, F);
```

Kode 3-7: Lastvektorene settes opp.

Krafttetthet-matrisen ble satt opp ved hjelp av variabelen q.

```
var Q = M.DenseDiagonal(Xnl.Count + 1, Xnl.Count + 1, q);
```

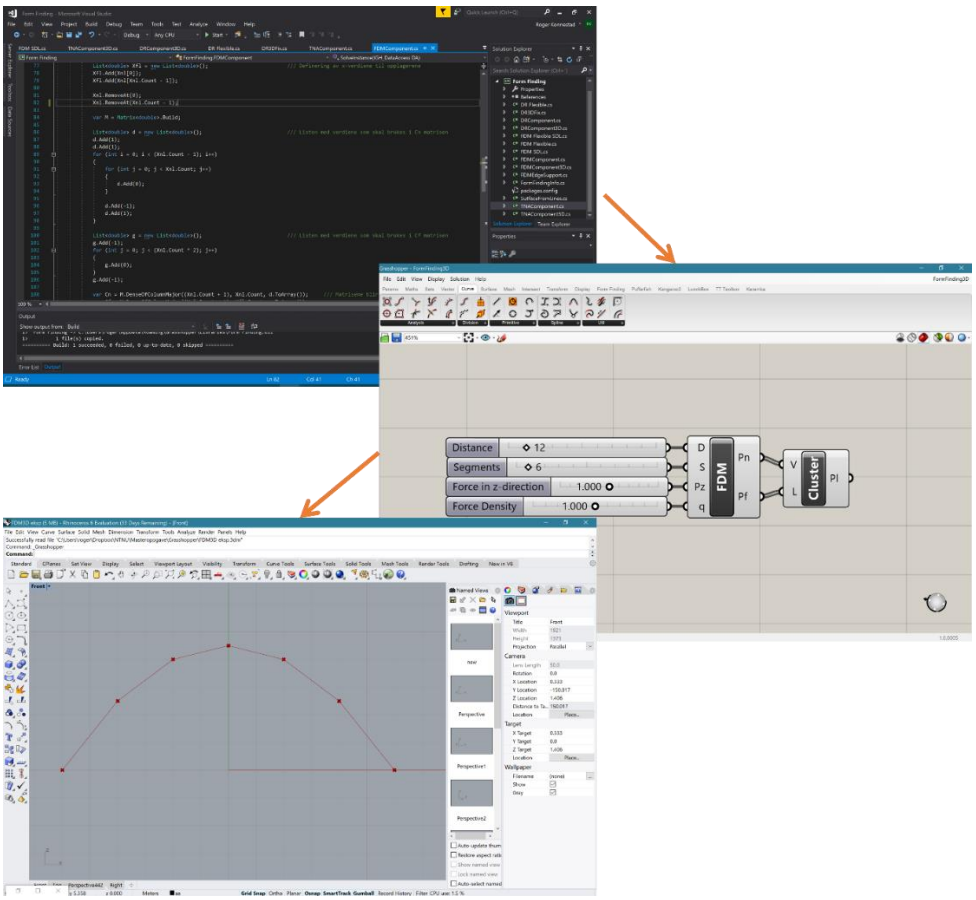
Kode 3-8: Krafttetthet-matrisen blir satt opp.

Når matrisene var satt opp måtte det gjøres utregninger for å få ut koordinatene til de nye punktene. Ut av dette kom de nye $\boldsymbol{x}$ og $\boldsymbol{y}$ matrisene som ble gjort om til GH punkter og utdata.

```
var CnT = (Cn.Transpose());
var Dn = CnT * Q * Cn;
var Df = CnT * Q * Cf;
var Dni = (Dn.Inverse());
var Xn = Dni * (Px - (Df * xf));
var Yn = Dni * (Py - (Df * yf));
```

Kode 3-9: Utregninger av $\boldsymbol{x}_n$ og $\boldsymbol{y}_n$.

Utdata ble da to lister med punkter. En for $\boldsymbol{x}_n$ og $\boldsymbol{y}_n$, og en for $\boldsymbol{x}_f$ og $\boldsymbol{y}_f$. Denne komponenten kunne da brukes i Grasshopper, og resultatene derfra kunne deretter visualiseres i Rhino. I Grasshopper ble det brukt «Number Slider» komponenter for å bestemme variablene som skulle være inndata i komponenten. Disse variablene kunne da enkelt endres på, ved å dra på skyveknappen, for å teste variablenes påvirkning på formen.



Figur 3-8: FDM, fra Visual Studio til Grasshopper til Rhino.

Måten lasten var lagt til på, med samme verdi i hvert punkt, ga en parabol-kurve som resultat. Dersom dette skulle vært for eksempel en bro med vaiere i punktene er dette ideelt, men hvis det skal være en selvbærende bue er ikke dette optimalt. For å finne den optimale formen i det tilfellet må lasten oppdateres etter formen. Dette vil igjen forandre formen som så igjen vil forandre lasten. Dette gjøres i flere iterasjoner til det konvergeres mot et resultat. Resultatet vil da være en kurvatur som vil være lik en kjedelinje.

```
for (int j = 0; j < it; j++)
{
    Yl = new List<double>();
    for (int i = 0; i < Yn.RowCount; i++)
    {
        Yl.Add(Yn.AsColumnMajorArray()[i]);
    }
    for (int i = 0; i < lXf; i++)
    {
        Yl.Add(0);
    }
    C = M.DenseOfColumnMajor((lXn + 1), lXn + lXf, e.ToArray());
    x = M.DenseOfColumnMajor(lXn + lXf, 1, Xl.ToArray());
    y = M.DenseOfColumnMajor(lXn + lXf, 1, Yl.ToArray());
    u = C * x;
    u1 = u.ToColumnMajorArray();
    U0 = M.DenseOfDiagonalArray(u1.Length, u1.Length, u1);
    v = C * y;
    v1 = v.ToColumnMajorArray();
    V0 = M.DenseOfDiagonalArray(v1.Length, v1.Length, v1);
    L0 = (U0.PointwisePower(2) + V0.PointwisePower(2)).PointwisePower(0.5);
    l0 = L0.Diagonal().ToColumnMatrix();
    Py = 0.5 * A * Cn.PointwiseAbs().Transpose() * l0;
    Yn = Dni * (Py - (Df * yf));
}
```

Kode 3-10: Iterasjoner der lasten oppdateres.

3.2.4 Thrust network analysis i 2D

Etter å ha laget en komponent for FDM var ikke overgangen til å gjøre det samme for TNA stor. TNA tar utgangspunkt i samme C-matrise og koordinatvektorer som FDM, så forskjellen var i selve utregningen. Krafttettheten fra FDM ble byttet ut med en skaleringsfaktor og en funksjon av grenlengdene. I stedet for å finne en Q-matrise basert på krafttetthet, regner man ut en T-matrise ved hjelp av horisontalt trykk l_h og grenlengdene L som igjen regnes ut ved hjelp av koordinatforskjellene U.

```
var e = new List<double>();
e.AddRange(d);
e.AddRange(g);
var C = M.DenseOfColumnMajor((lXn + 1), lXn + lXf, e.ToArray());
```

Kode 3-11: Den fullstendige C-matrisen blir satt opp.

```
var Xl = new List<double>();
Xl.AddRange(Xnl);
Xl.AddRange(Xfl);
var x = M.DenseOfColumnMajor(lXn + lXf, 1, Xl.ToArray());
```

Kode 3-12: Den fullstendige x-matrisen blir satt opp.

```
var u = C * x;
var u1 = u.ToColumnMajorArray();
var u2 = u.RowCount;
for (int i = 0; i < u2; i++)
{
    u1[i] = Math.Abs(u1[i]);
}
var U = M.DenseOfDiagonalArray(lXn + 1, lXn + 1, u1);
var lh = M.Dense(lXn + 1, 1, u1[0]);
var t = (U.Inverse()) * lh;
var T = M.DenseOfDiagonalArray(lXn + 1, lXn + 1, t.ToColumnMajorArray());
```

Kode 3-13: Utregning av T-matrisen.

3.2.5 Dynamic relaxation i 2D

Dynamic relaxation likner ikke like mye på FDM som TNA, men det er flere likheter. C-matrisen settes opp på samme måte, og utregningen av D_n er veldig lik. Men i stedet for å bruke krafttetthet i Q-matrisen brukes mer intuitive verdier som EA. I tillegg er de indre kreftene en egen variabel.

```
var f0 = M.Dense(lXn + 1, 1, f0n);
var EAV = M.Dense(lXn + 1, 1, EA);
var e = (L - L0) * (l0.PointwisePower(-1));
```

Kode 3-14: Oppsett av vektorene til de indre kreftene, EA og e.

I DR brukes Newtons 2. lov og man regner ut fart og akselerasjon som følge av belastingen. Man vil da få en bevegelse som vil svinge rundt en stabil løsning. Viskøs demping brukes for å få konvergens.

```
var dt = 1;
var m = 0.5*dt*Cn.Transpose().PointwiseAbs()*(L.Inverse()*f0 + L0.Inverse()*EAV);
var M = M.DenseOfDiagonalArray(m.Column(0).AsArray());
var C = 0.5;
var A = (1 - C / 2) / (1 + C / 2);
var B = (1 + A) / 2;
var v = M.Dense(lXn, 1, 0);
```

Kode 3-15: Oppsett av matriser og vektorer som skal brukes for å finne fart og nye koordinater.

```
var v1 = A0 * v0 + B0 * dt * M0.Inverse() * ry;
var y1 = yn + dt * v1;
var v11 = A0 * v0 + B0 * dt * M0.Inverse() * rx;
var x1 = xn + dt * v11;
```

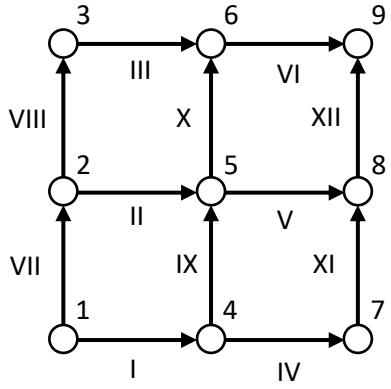
Kode 3-16: Utregning av fart og koordinater.

Med disse nye koordinatene må vi oppdatere L-matrisen og alle matriser og vektorer som bygger på den i tillegg til de tilfellene der koordinatene brukes direkte. Fart-vektorene må også oppdateres ved å sette inn de nye fart-verdiene. Dette gjøres ved å sett opp en stor løkke som går over et antall iterasjoner. Antall iterasjoner ble i programmet satt som en inndata-parameter som kunne bestemmes manuelt. Etter hvert som antall iterasjoner økes svinger konstruksjonen frem og tilbake rundt en løsning som den går mot på grunn av dempingen som er lagt inn. Hvor mange iterasjoner som trengs må vurderes i hvert enkelt tilfelle. Hvis konstruksjon ikke er for kompleks kan man raskt se, ved å endre på parameteren, når den konvergerer.

3.2.6 Eksempel 2: Formfinning i 3D

Etter å ha utviklet 2D komponenter, ble det utviklet komponenter i 3D med bruk av de samme metodene som i foregående delkapitler. De første 3D-komponentene tok utgangspunkt i et rektangulært rutenett i x-y planet. Det kunne være forskjellig antall linjer og noder i x- og y-retning og avstanden mellom nodene kunne også variere.

Under vises utregning av et eksempel ved hjelp av Force density method. Geometrien består av 9 noder og 12 linjer med avstand 1 mellom hver node og belastning lik 1 i negativ z-retning i hvert punkt:



Figur 3-9: Utgangspunktet for formfinning i 3D.

Definering $\boldsymbol{C}$-matrisen og koordinat-vektorene $\boldsymbol{x}$ og $\boldsymbol{y}$:

$$\boldsymbol{C} = [\boldsymbol{C}_N \quad \boldsymbol{C}_F] = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{x} = [\boldsymbol{x}_N \quad \boldsymbol{x}_F] = [1 \quad 0 \quad 1 \quad 2 \quad 1 \quad 0 \quad 2 \quad 0 \quad 2]^T$$

$$\boldsymbol{y} = [\boldsymbol{y}_N \quad \boldsymbol{y}_F] = [0 \quad 1 \quad 1 \quad 1 \quad 2 \quad 0 \quad 0 \quad 2 \quad 2]^T$$

$$\boldsymbol{z} = [\boldsymbol{z}_N \quad \boldsymbol{z}_F] = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$$

Koordinat forskjellene satt opp som $\boldsymbol{u}$ og $\boldsymbol{v}$:

$$\boldsymbol{u} = \boldsymbol{Cx}, \boldsymbol{v} = \boldsymbol{Cy}, \boldsymbol{w} = \boldsymbol{Cz}$$

$$\boldsymbol{u} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 2 \\ 1 \\ 0 \\ 2 \\ 0 \\ 2 \end{bmatrix}$$

$$= [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]^T$$

$$\boldsymbol{v} = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$$

$$\boldsymbol{w} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$$

$\boldsymbol{u}$, $\boldsymbol{v}$ og $\boldsymbol{w}$ er diagonalene av $\boldsymbol{U}$, $\boldsymbol{V}$ og $\boldsymbol{W}$ som brukes til å finne gren-lengdene $\boldsymbol{L}$ som har diagonalen $\boldsymbol{l}$:

$$\boldsymbol{L} = (\boldsymbol{U}^2 + \boldsymbol{V}^2 + \boldsymbol{W}^2)^{\frac{1}{2}}$$

$$\boldsymbol{l} = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]^T$$

Nå er det klart for å ta i bruk Force density method:

Ytre last defineres:

$$\boldsymbol{p}_x = 0$$

$$\boldsymbol{p}_y = 0$$

$$\boldsymbol{p}_z = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]^T$$

Krafttettheten defineres:

$$\boldsymbol{q} = \boldsymbol{L}^{-1}\boldsymbol{f} = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]^T$$

$\boldsymbol{D_N}$, og $\boldsymbol{D_F}$ regnes ut på samme måte som før:

$$\boldsymbol{D_N} = \boldsymbol{C_N^T Q C_N}$$

$$\boldsymbol{D_F} = \boldsymbol{C_N^T Q C_F}$$

Utregning av de nye koordinatene $\boldsymbol{x_N}$, $\boldsymbol{y_N}$ og $\boldsymbol{z_N}$ på samme måte som i 2D eksemplet:

$$\boldsymbol{x_N} = \boldsymbol{D_N^{-1}}(\boldsymbol{p_x} - \boldsymbol{D_F x_F})$$

$$\boldsymbol{y_N} = \boldsymbol{D_N^{-1}}(\boldsymbol{p_y} - \boldsymbol{D_F y_F})$$

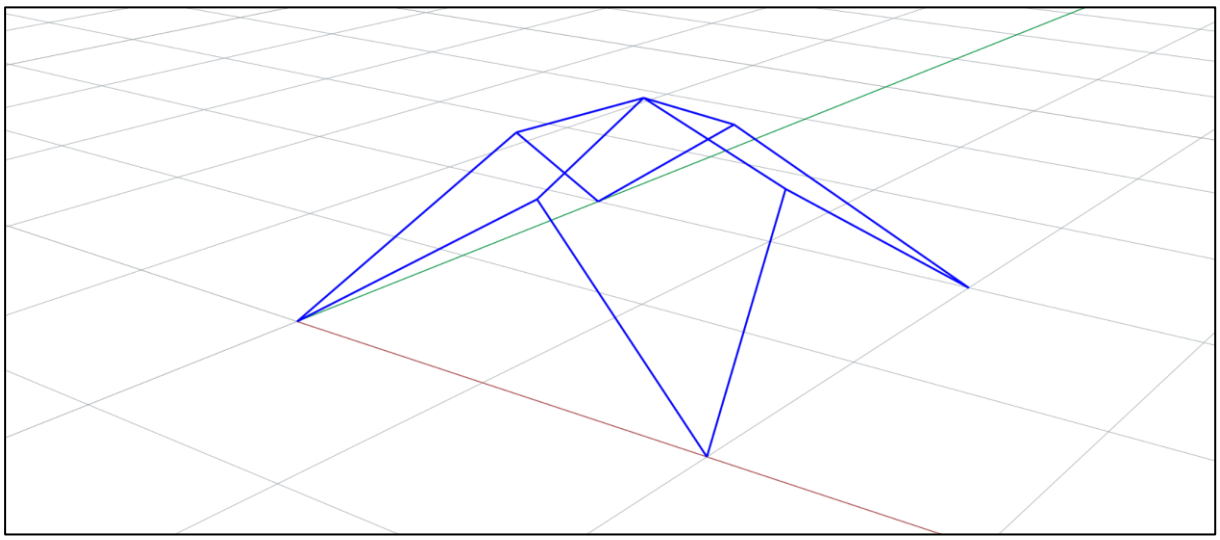
$$\boldsymbol{z_N} = \boldsymbol{D_N^{-1}}(\boldsymbol{p_z} - \boldsymbol{D_F y_F})$$

$$\boldsymbol{x_N} = [0{,}333 \quad 1 \quad 1 \quad 1 \quad 1{,}667]^T$$

$$\boldsymbol{y_N} = [1 \quad 0{,}333 \quad 1 \quad 1{,}667 \quad 1]^T$$

$$\boldsymbol{z_N} = [0{,}625 \quad 0{,}625 \quad 0{,}875 \quad 0{,}625 \quad 0{,}625]^T$$

Resultatet modellert:



3.2.7 Force density method i 3D

Når FDM skulle utvikles i 3D var mye likt som i 2D. Fortsatt var krafttetthet og belastning inndata, men i stedet for å ha en linje som utgangspunkt var det nå et rutenett. Inndata ble nå en liste med punkter, sortert på samme måte som i 3.2.6 Eksempel 2: Formfinning i 3D. Dette medførte at det å sette opp $\boldsymbol{C}$-matrisen ble enda mer komplisert enn for 2D eksempelet. Komponenten skulle ha fire opplagere, et i hvert hjørne. Det ble i komponenten laget en parameter for verdien til lasten i z-retning. I x- og y-retning ble verdien satt lik 0. Utregningen av matrisene var lik som for 2D eksemplet og utdata ble en liste med punkter med nye koordinater. Antall inndelinger i U og V retning ble også valgt som inndata.

```
List<double> x = new List<double>();
for (int i = 0; i < points.Count; i++)
{
    x.Add(points[i].X);
}
var X = M.DenseOfColumnMajor(x.Count, 1, x.ToArray());

List<double> y = new List<double>();
for (int i = 0; i < points.Count; i++)
{
y.Add(points[i].Y);
}
var Y = M.DenseOfColumnMajor(y.Count, 1, y.ToArray());

List<double> z = new List<double>();
for (int i = 0; i < points.Count; i++)
{
    z.Add(points[i].Z);
}
var Z = M.DenseOfColumnMajor(z.Count, 1, z.ToArray());
```

Kode 3-17: $\boldsymbol{x}$, $\boldsymbol{y}$ og $\boldsymbol{z}$-matriser settes opp fra punktene.

For å kunne sette opp $\boldsymbol{C}$-matrisen med riktige dimensjoner ble U og V verdiene tatt i bruk.

```
var u = U * (V + 1);
var v = V * (U + 1);
var n = (V + 1) * (U + 1);
```

Kode 3-18: Utregning av u: øverste rader av $\boldsymbol{C}$-matrisen, v: nederste rader av $\boldsymbol{C}$-matrisen og n: antall punkter.

Under vises hvordan x_n og x_f ble regnet ut. Tilsvarende ble gjort for y_n, y_f, z_n og z_f.

```
var Xn1 = X.RemoveRow(n - 1);
var Xn2 = Xn1.RemoveRow(n - V - 1);
var Xn3 = Xn2.RemoveRow(V);
var Xn = Xn3.RemoveRow(0);

var Xf1 = X.Row(0);
var Xf2 = X.Row(V);
var Xf3 = X.Row(n - V - 1);
var Xf4 = X.Row(n - 1);
var Xf = M.DenseOfRowVectors(Xf1, Xf2, Xf3, Xf4);
```

Kode 3-19: Utgrening av x_n og x_f.

Oppsettet av C-matrisen ble i dette tilfellet gjort på en komplisert måte. På samme måte som for 2D-komponenten var utgangspunktet mønsteret som formet seg i C-matrisen.

```
List<double> g = new List<double>();
List<double> h = new List<double>();
for (int i = 0; i <= n; i++)
{
    g.Add((V + 1) * i);
    h.Add(((V + 1) * i) + 1);
}

            List<double> I = new List<double>(); /// Liste over verdiene i C-matrisen
            for (int i = 1; i <= n; i++)
            {
                if (i > (V + 2))
                {
                    for (int j = 1; j <= (i - V - 2); j++)
                    {
                        I.Add(0);
                    }
                }
                if (i > (V + 1))
                {
                    I.Add(1);
                }
                if (i <= V)
                {
                    for (int j = 1; j <= (i - 1); j++)
                    {
                        I.Add(0);
                    }
                }
                if (i > V)
                {
                    for (int j = 1; j <= V; j++)
                    {
                        if (i <= (u + 1))
                        {
                            I.Add(0);
                        }
                    }
```

```
            }
            if (i > u + 1)
            {
                for (int j = 1; j <= (u + V + 1 - i); j++)
                {
                    if (i < n)
                    {
                        I.Add(0);
                    }
                }
            }
            if (i <= u)
            {
                I.Add(-1);
            }
            if (i < u)
            {
                for (int j = 1; j <= (u - i); j++)
                {
                    I.Add(0);
                }
            }
            //
            for (int j = 1; j <= (i - 2); j++)
            {
                I.Add(0);
            }
            for (int j = 2; j <= (i - 1); j++)
            {
                if (h.Contains(j))
                {
                    var c = I.Count;
                    I.RemoveAt(c - 1);
                }
            }
            if (h.Contains(i))
            {

            }
            else
            {
                I.Add(1);
            }
            if (g.Contains(i))
            {
                if (i < n - 1)
                {
                    I.Add(0);
                }
            }
            else
            {
                I.Add(-1);
            }
            if (h.Contains(i))
            {
                if (i < n - 1)
                {
                    I.Add(0);
                }
            }
            for (int j = 1; j <= v - i; j++)
```

```
            {
                I.Add(0);
            }
            for (int j = 2; j <= (i - 1); j++)
            {
                if (h.Contains(j))
                {
                    if (i < (n - 1))
                    {
                        I.Add(0);
                    }
                }
            }
            if (i >= (n - U))
            {
                if (i < (n - 1))
                {
                    for (int k = 0; k <= (i - n + U); k++)
                    {
                        var c = I.Count;
                        I.RemoveAt(c - 1);
                    }
                }
            }
            if (i == 1)
            {
                var c = I.Count;
                I.RemoveAt(c - 1);
            }
        }

        var C = M.DenseOfColumnMajor((u + v), n, I.ToArray());
```

Kode 3-20: Oppsett av C-matrisen.

Deling av C-matrisen inn i C_n og C_f ble gjort på samme måte som x_n og x_f for x. Lastvektor, krafttetthet-matrisen og utregningene av de nye koordinatene ble gjort på samme måte som for 2D-eksemplet. Utdata ble tre lister med punkter. En for x_n, y_n og z_n, en for x_f, y_f og z_f, og en for alle punktene.

3.2.8 Thrust network analysis i 3D

Etter å ha laget FDM i 3D og TNA i 2D var det ikke mye nytt som måtte programmeres for å lage TNA i 3D. På samme måte som for 2D eksemplet, bortsett fra hvordan $\boldsymbol{T}$-matrisen ble satt opp, var det bare å ta litt fra begge komponentene og sette sammen til den nye. For å sette opp $\boldsymbol{T}$-matrisen måtte grenlengdene regnes ut i alle tre akseretninger og settes sammen i motsetning til i 2D der bare x-retningen ble brukt. Dermed måtte også koordinatforskjellene $\boldsymbol{U}$, $\boldsymbol{V}$ og $\boldsymbol{W}$ regnes ut i y- og z-retning.

```
            var u0 = C * X;
            var u1 = u0.ToColumnMajorArray();
            var u2 = u0.RowCount;
            for (int i = 0; i < u2; i++)
            {
                u1[i] = Math.Abs(u1[i]);
            }
            var U0 = M.DenseOfDiagonalArray(u1.Length, u1.Length, u1);

            var v0 = C * Y;
            var v1 = v0.ToColumnMajorArray();
            var v2 = v0.RowCount;
            for (int i = 0; i < v2; i++)
            {
                v1[i] = Math.Abs(v1[i]);
            }
            var V0 = M.DenseOfDiagonalArray(v1.Length, v1.Length, v1);

            var w0 = C * Z;
            var w1 = w0.ToColumnMajorArray();
            var w2 = w0.RowCount;
            for (int i = 0; i < w2; i++)
            {
                w1[i] = Math.Abs(w1[i]);
            }
            var W0 = M.DenseOfDiagonalArray(w1.Length, w1.Length, w1);

            var Lh = (U0.PointwisePower(2) + V0.PointwisePower(2) +
W0.PointwisePower(2)).PointwisePower(0.5);
            var lh = M.Dense(u1.Length, 1, Lh.At(0,0));
            var t = Lh * lh;
            var T = M.DenseOfDiagonalArray(u + v, u + v, t.ToColumnMajorArray());
```

Kode 3-21: Utregning av $\boldsymbol{T}$-matrisen.

3.2.9 Dynamic relaxation i 3D

Etter å ha satt opp DR i 2D var det bare å gjøre tilsvarende i x og y retning som var gjort i z-retning. I tillegg til $\boldsymbol{U}$ og $\boldsymbol{V}$ måtte $\boldsymbol{W}$ regnes ut for å finne $\boldsymbol{L}$. Akselerasjon, fart og nye koordinater måtte også regnes ut i x- og y-retning og alle måtte oppdateres for hver iterasjon.

3.2.10 Utbedrede komponenter

Etter at alle komponentene var laget i 3D ble neste skritt å gjøre komponentene mer fleksible. Slik komponentene hadde blitt utviklet var de veldig begrenset i bruksområde. De kunne bare ta utgangspunkt i rektangulære flater og rutenettet kunne bare være kvadratisk. For å gjøre komponentene mer nyttige for design av skallkonstruksjoner var det da to ting som måtte legges til; mulighet for å ta utgangspunkt i en hvilken som helst flat form, og å ha et hvilket som helst nettverk av linjer inne i flaten.

Utfordringen i å lage disse nye komponentene er hvordan C-matrisen skal settes opp. Komponentene som hadde blir laget tidligere tok utgangspunkt i mønsteret i C-matrisen, dette var en unødvendig komplisert måte å lage komponenten i 3D på. De nye komponentene skulle utvikles på en helt annen langt mer fleksibel måte.

Inndata til disse nye komponentene skulle være linjer i stedet for noder. Det skulle være to inndata en for linjene som skulle virke som opplagere (Le) og en for linjene som skulle kunne oppdateres i komponenten (Li). Ut i fra disse linjene kunne alle nodene (P) bli funnet.

```
List<Point3d> P = new List<Point3d>();
for (int i = 0; i < Li.Count; i++)
{
    P.Add(Li[i].PointAt(0));
    P.Add(Li[i].PointAt(1));
}
for (int i = 0; i < Le.Count; i++)
{
    P.Add(Le[i].PointAt(0));
    P.Add(Le[i].PointAt(1));
}
List<Point3d> P0R = new List<Point3d>();
for (int i = 0; i < P.Count; i++)
{
    var PRc0 = P0R.Count;
    for (int j = i + 1; j < P.Count; j++)
    {
        var PRc1 = P0R.Count;
        if (P[i].Equals(P[j]))
        {
            if (PRc0 == PRc1)
            {
                P0R.Add(P[j]);
            }
        }
    }
}
for (int i = 0; i < P0R.Count; i++)
{
    P.Remove(P0R[i]);
}
```

Kode 3-22: Liste med alle punktene blir opprettet og sortert slik at opplagerpunktene kommer sist i listen.

Neste skritt var å sette opp C-matrisen. Det første som ble gjort var å sette opp en tom C-matrise med riktige dimensjoner. Antall kolonner i matrisen er likt antall punkter, og rekkefølgen følger listen med punkter som ble laget over. For å finne posisjonen til startpunktene og endepunktene i matrisen, ble det laget løkker som søkte etter når linjene hadde startpunkter og endepunkter som var de samme punktene i punktlisten. Når punktet ble funnet byttet koden verdien i matrisen til -1 eller +1 dersom den fant samme punkt i punktlisten.

```
            var C = M.Dense(Le.Count + Li.Count, P.Count, 0); /// C-matrise settes opp

            for (int i = Le.Count; i < (Le.Count + Li.Count); i++)        /// Cn delen
            {
                for (int j = 0; j < P.Count; j++)
                {
                    if (Li[i-Le.Count].PointAt(0) == P[j])
                    {
                        C[i, j] = -1;       /// Ved startpunktet settes verdien lik -1
                    }
                    if (Li[i-Le.Count].PointAt(1) == P[j])
                    {
                        C[i, j] = 1;         /// Ved endepunktet settes verdien lik +1
                    }
                }
            }
            for (int i = 0; i < Le.Count; i++)
            {
                for (int j = 0; j < P.Count; j++)                         /// Cf delen
                {
                    if (Le[i].PointAt(0) == P[j])
                    {
                        C[i, j] = -1;       /// Ved startpunktet settes verdien lik -1
                    }
                    if (Le[i].PointAt(1) == P[j])
                    {
                        C[i, j] = 1;         /// Ved endepunktet settes verdien lik +1
                    }
                }
            }
```

Kode 3-23: C-matrisen regnes ut.

Resten av utregningene ble utført på samme måte som for tidlige komponenter. Det ble først laget en forbedret DR og deretter en for FDM for å sammenlikne. Med disse komponentene kan man definere mange forskjellige former på flere forskjellige måter. Man kan for eksempel tegne en hvilken som helst lukket polylinje, som definerer et areal, og inne i kurven kan man så lage et mesh. Når man har et mesh kan man bruke en komponent i Grasshopper som heter «Mesh Edges». Med denne komponenten finner man alle linjene i meshet og de blir sortert etter linjene langs kanten og linjene inne i meshet. Disse linjene brukes da som utgangspunkt for de nye komponentene ved å dra en linje i Grasshopper fra «Mesh Edges» over i de nye komponentene. Man kan også tegne linjene manuelt i Rhino eller Grasshopper og legge dem til direkte i komponenten.

For å gjøre det mulig å utforme konstruksjoner med åpninger ble det lagt til en ekstra parameter som gjør dette mulig. Man kan med komponenten som nå er utviklet lage en hvilken som helst lukket konstruksjon der det er opplagere langs hele kanten. Dersom man vil ha en eller flere åpninger må det gjøres en justering i hvordan $\boldsymbol{C}$-matrisen og koordinatvektorene deles opp. Det ble definert en egen parameter (O) for antall åpninger konstruksjonen skulle ha. Under vises hvordan $\boldsymbol{x_n}$ og $\boldsymbol{x_f}$ blir satt opp med denne nye parameteren.

```
List<double> x = new List<double>();
for (int i = 0; i < P.Count; i++)
{
    x.Add(P[i].X);
}
List<double> Vxnl = new List<double>();
for (int i = 0; i < (P.Count - Le.Count - O); i++)
{
    Vxnl.Add(x[i]);
}
var Xn = M.DenseOfColumnMajor(Vxnl.Count, 1, Vxnl.ToArray());
List<double> Vxfl = new List<double>();
for (int i = (P.Count - Le.Count - O); i < P.Count; i++)
{
    Vxfl.Add(x[i]);
}
var Xf = M.DenseOfColumnMajor(Vxfl.Count, 1, Vxfl.ToArray());
```

Kode 3-24: $\boldsymbol{x_n}$ og $\boldsymbol{x_f}$ settes opp ut i fra punktene med O som variabel for antall åpninger i konstruksjonen.

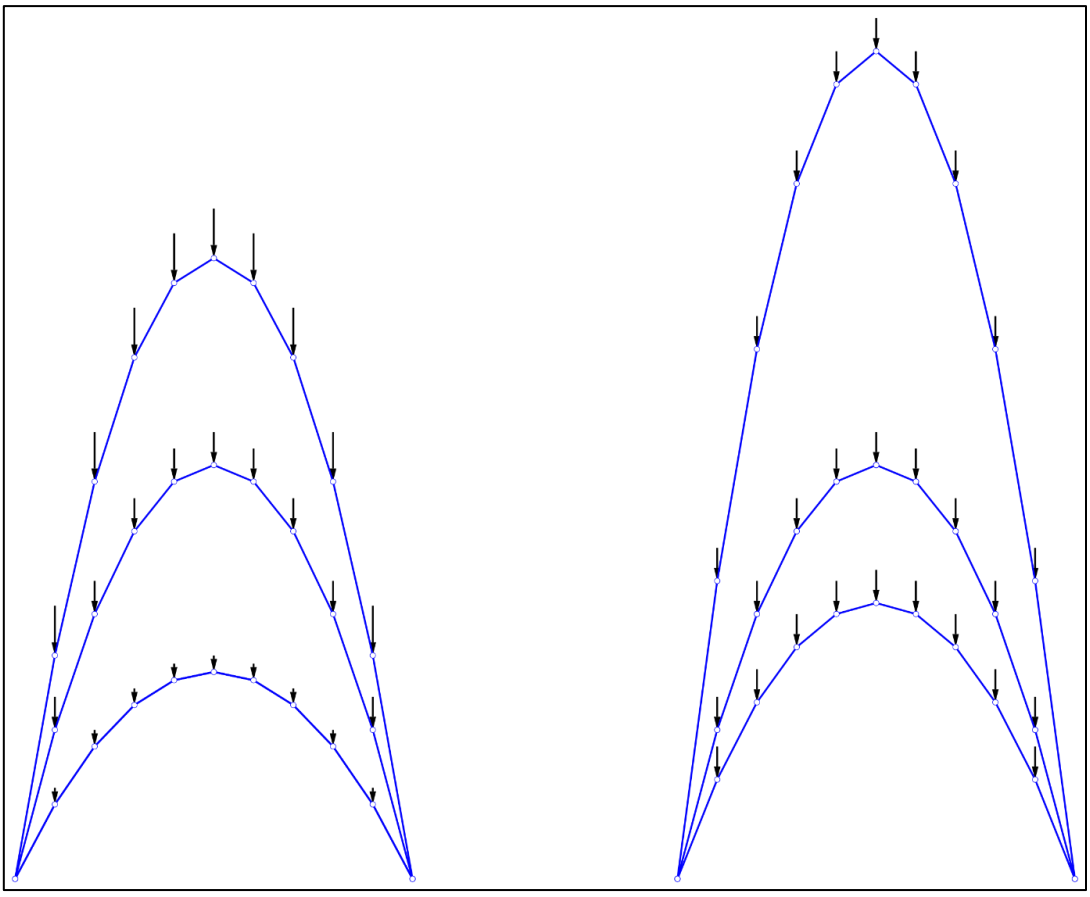
4 Resultat og analyse

Testing og sammenlikning av komponentene. Resultatene vil til slutt materialiseres ved hjelp av strukturanalyser.

4.1 Testing av komponentene

4.1.1 Force density method i 2D

Force density method-komponenten som ble utviklet gjorde det mulig å lage buer basert på flere parametere. I tillegg til lenge og antall segmenter, ble belastning og krafttetthet definert som parametere man raskt og enkelt kunne endre på. Under vises flere eksempler på hvordan formen blir ved ulike verdier for belastning og krafttetthet.

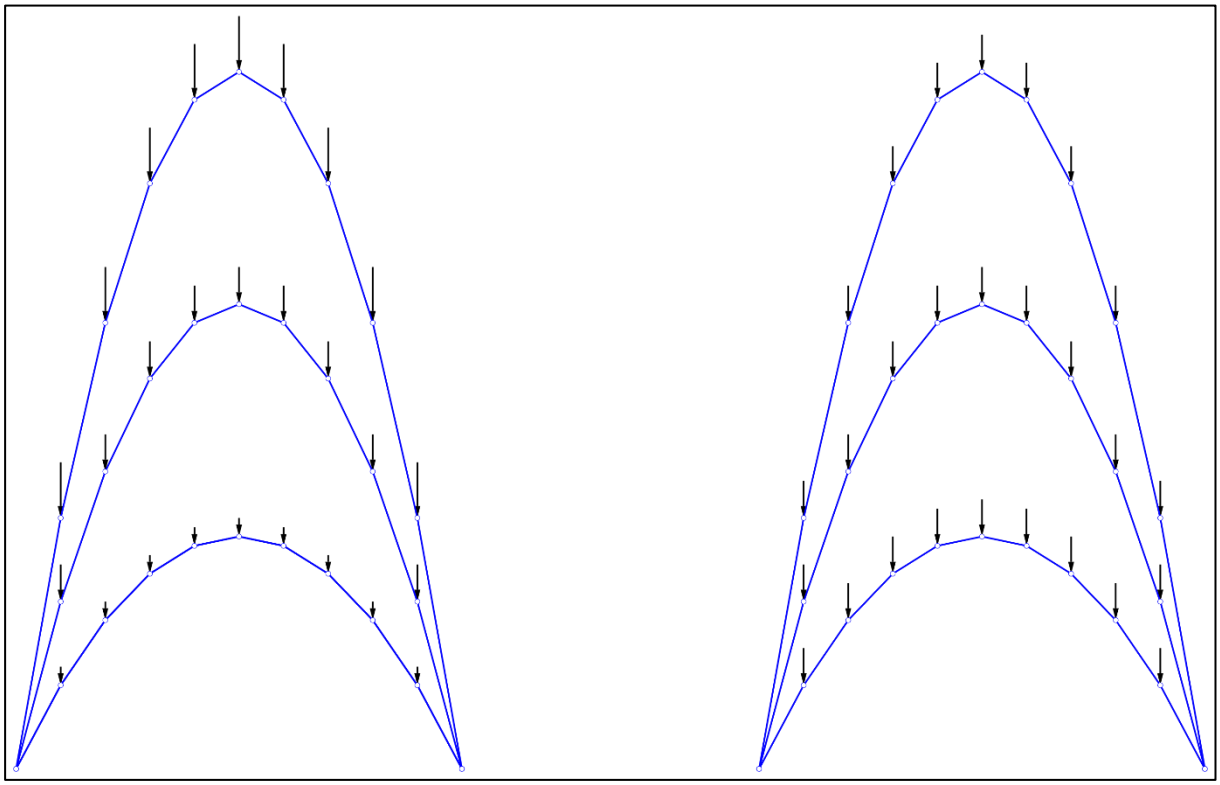


Figur 4-1: Til venstre: FDM med krafttetthet lik 1 på alle og belastning lik 1,5 (øverste), 1 (den i midten) og 0,5 (nederste). Til høyre: FDM med belasting lik 1 på alle og krafttetthet lik 1,5 (nederste), 1 (den i midten) og 0,5 (øverste). Figuren er laget i Rhino.

Som resultatet viser får vi en brattere konstruksjon ved større belastninger. På samme måte som en kjedelinje vil trekke lengre ned ved tyngre laster, vil denne buen trekkes oppover ved tyngre laster. Krafttetthet sier noe om hvor stor belastning hver stav har mulighet til å ta opp per lengdeenhet. Med stor krafttetthet vil man da få en bue som ikke blir så bratt som en med mindre krafttetthet.

4.1.2 Thrust network analysis i 2D

Thrust network analysis-komponenten som ble utviklet likner ganske mye på FDM komponenten. Alle parameterne var like bortsett fra at krafttetthet var byttet ut med en skaleringsfaktor. Under vises flere eksempler på hvordan formen blir ved ulike verdier på belastningen og skaleringen.

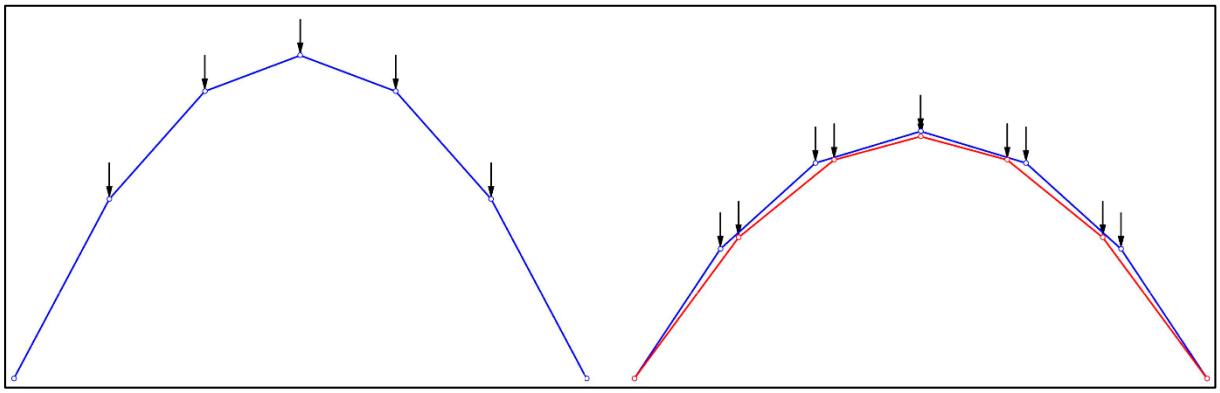


Figur 4-2: Til venstre: TNA med skalering lik 1 på alle og belastning lik 1,5 (øverste), 1 (den i midten) og 0,5 (nederste). Til høyre: TNA med belasting lik 1 på alle og skalering lik 1,5 (øverste), 1 (den i midten) og 0,5 (nederste). Figuren er laget i Rhino.

Som vi ser på figuren til venstre fungerer lastparameteren på samme måte her som i FDM-komponenten. Skaleringsfaktoren fungerer slik at med en stor verdi skaleres buen til en større og brattere kurve og med en liten faktor blir det en mindre og slakere kurve.

4.1.3 Dynamic relaxation i 2D

Dynamic relaxation-komponenten som ble utviklet fungerte litt annerledes enn de andre. Den hadde ikke krafttetthet eller en skaleringsfaktor som parametere, men aksialstivhet og indre krefter. I tillegg gir ikke DR et ferdig resultat med en gang, men den trenger flere iterasjoner for å konvergere fram til en løsning. Under vises flere eksempler på hvordan formen blir ved ulike verdier på belastningen, aksialstivhet og indre krefter.



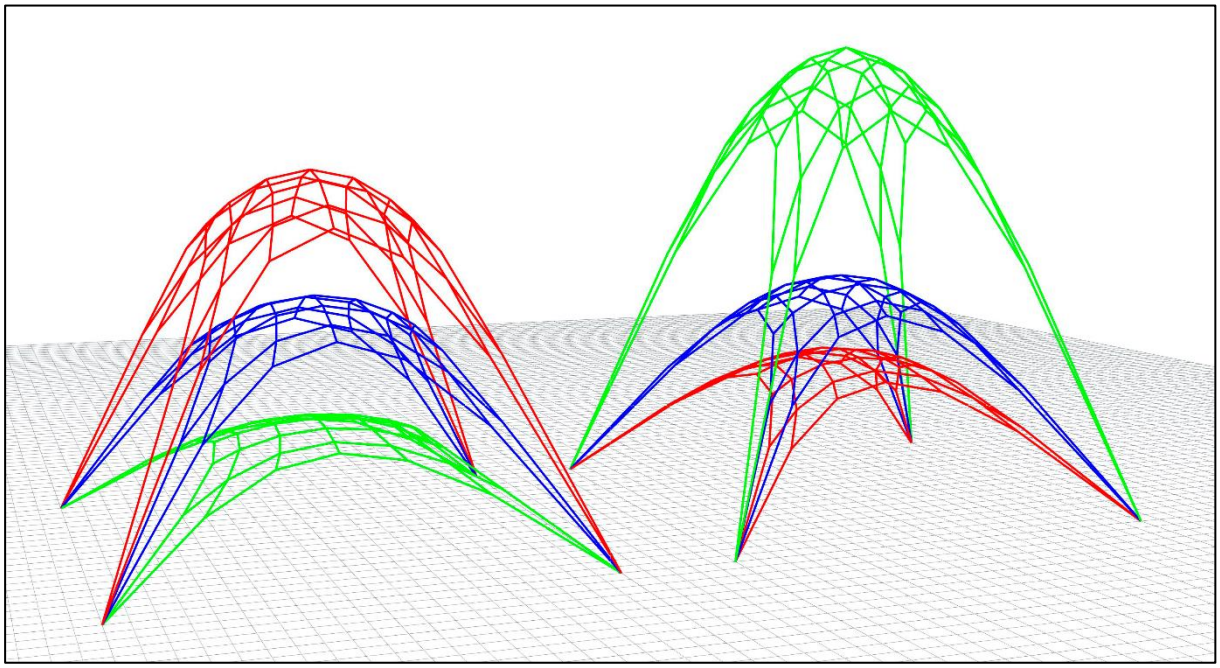
Figur 4-3: Til venstre: DR med belastning lik 0.75, EA lik 1 og indre krefter lik 1.
Til høyre (blå): DR med belastning lik 0.75, EA lik 2 og indre krefter lik 1.
Til høyre (rød): DR med belastning lik 0.75, EA lik 1 og indre krefter lik 1.5.
Figuren er laget i Rhino.

Ut i fra figuren over ser vi at aksialstivhet og indre krefter påvirker buen på en annen måte enn belastningen. Ved å bare endre på belastningen til figuren til venstre vil man ikke klare å finne noen av de formene til høyre som er funnet ved å endre på aksialstivhet og indre krefter. Det man vil finne er en bue som legger seg mellom de to som er funnet. Den blå buen med stor aksialstivhet vil ha lengre linjer ved toppen, mens den røde med store indre krefter vil ha kortere linjer ved toppen.

Med større aksialstivhet vil man kunne ha lengre linjer i områder med større momenter. Store indre krefter med mindre aksialstivhet gir en form som har kortere linjer der momentet er stort. Dette for å få en form som har mindre momenter.

4.1.4 Force density method i 3D

Inndata til Force density method-komponenten i 3D var en liste med punkter organisert over et kvadratisk rutenett, antall rader og kolonner i rutenettet, krafttetthet og belastning. På figuren under vises en sammenlikning av forskjellige resultater med forskjellige inndata. Utgangspunktet var et kvadrat på 20x20 enheter med 6 oppdelinger i U og V retning.



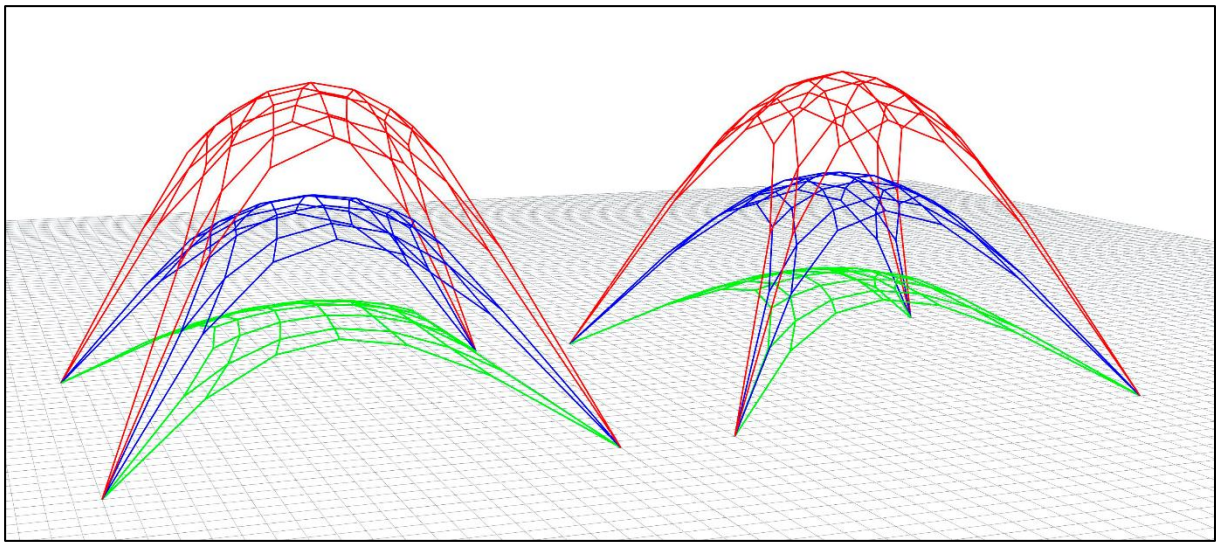
Figur 4-4: Til venstre: FDM med krafttetthet lik 1 på alle og belastning lik 1,5 (øverste), 1 (den i midten) og 0,5 (nederste). Til høyre: FDM med belasting lik 1 på alle og krafttetthet lik 1,5 (nederste), 1 (den i midten) og 0,5 (øverste). Figuren er laget i Rhino.

Ut i fra figuren ser man at med større belastning får man en brattere konstruksjon, mens med mindre belastning får man en flatere konstruksjon. Siden det må være likevekt mellom ytre og indre krefter blir de indre kreftene større når den ytre lasten øker. Når krafttetthet bestemmer forholdet mellom indre krefter og stavlengder vil stavlengdene bli lengre når den ytre kraften blir større og siden opplagerne er bestemt vil det føre til at konstruksjonen blir brattere.

Når krafttettheten økes betyr det at hver stav kan ta opp en større last per lengdeenhet. Når krafttettheten økes uten at ytre last endres, vil det føre til at stavlengdene blir kortere og konstruksjonene dermed flatere.

4.1.5 Thrust newtork analysis i 3D

Inndata til Thrust network analysis-komponenten i 3D var en liste med punkter organisert over et kvadratisk rutenett, antall rader og kolonner i rutenettet, en skaleringsfaktor og belastning. På figuren under vises en sammenlikning av forskjellige resultater med forskjellige inndata.

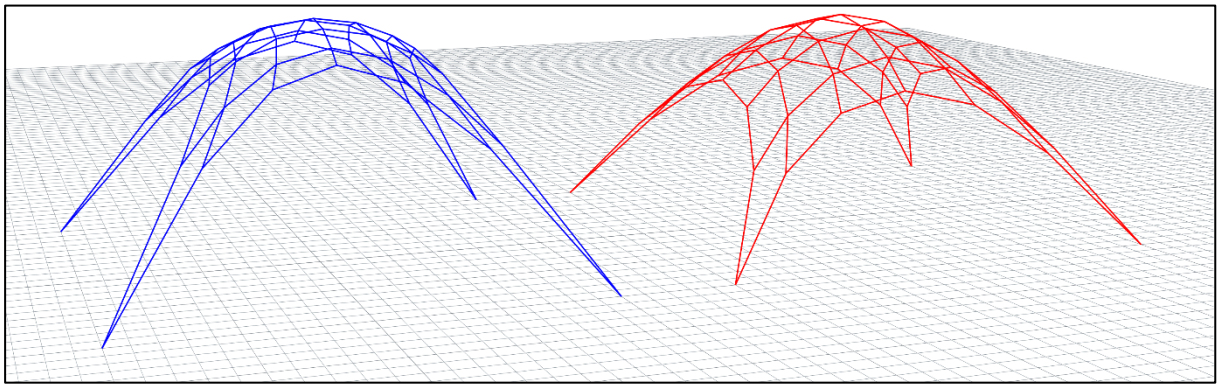


Figur 4-5: Til venstre: TNA med skalering lik 10 på alle og belastning lik 1,5 (øverste), 1 (den i midten) og 0,5 (nederste). Til høyre: TNA med belasting lik 1 på alle og skalering lik 15 (øverste), 10 (den i midten) og 5 (nederste). Figuren er laget i Rhino.

Ytre last påvirker konstruksjonen på samme måte om i FDM. Med større last får man en brattere konstruksjon. Ved å endre på skaleringsfaktoren skaleres konstruksjonen.

4.1.6 Dynamic relaxation i 3D

Inndata til Dynamic relaxation-komponenten i 3D var en liste med punkter organisert over et kvadratisk rutenett, antall rader og kolonner i rutenettet, en aksialstivhet og belastning. På figuren under vises en sammenlikning av forskjellige resultater med forskjellige inndata.



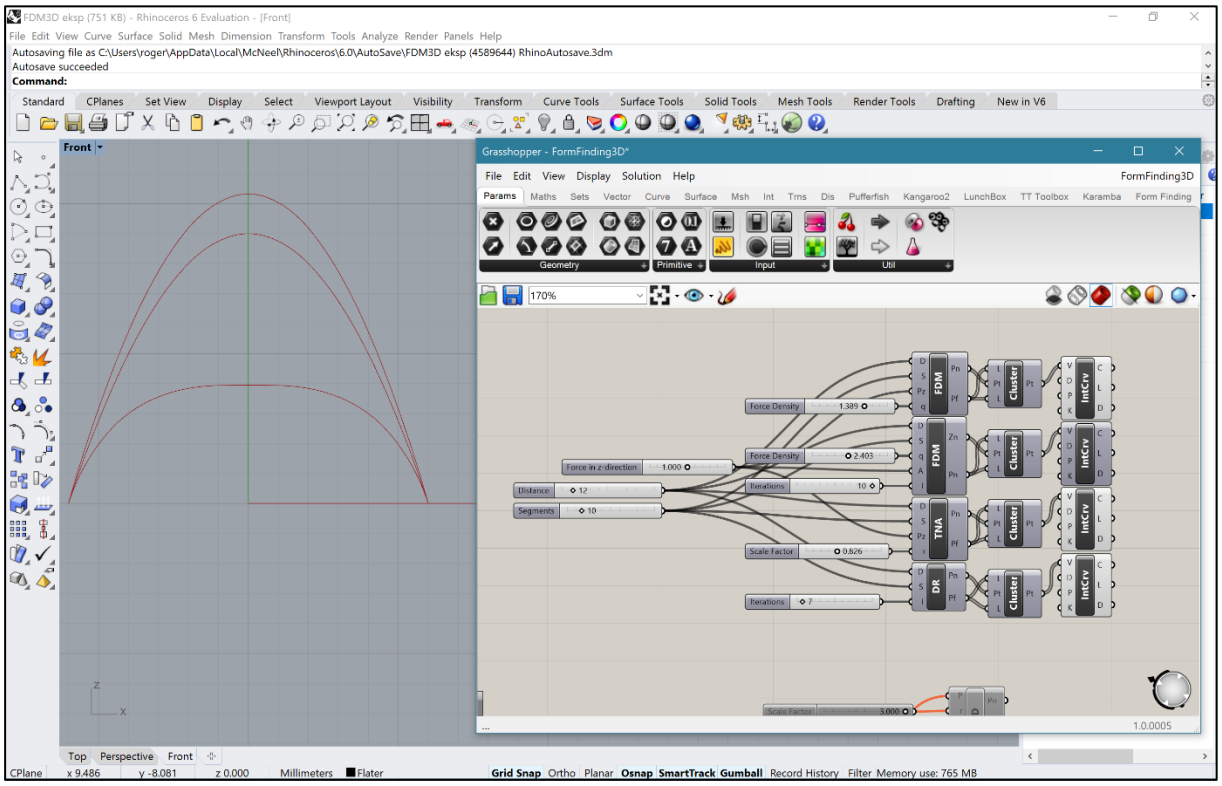
Figur 4-6: Til venstre: DR med belastning lik 0,3 og indre krefter lik 1 på begge og EA lik 1 (venstre) og 2 (høyre). Figuren er laget i Rhino.

På samme måte som for 2D-komponenten ser vi at med større aksialstivhet får vi lengre linjer i toppen av konstruksjonen og kortene langs opplagerne. Dette fører til at man ender opp med en konstruksjon der linjelengdene varierer i litt mindre grad og man får en mindre bratt kurvatur.

4.2 Sammenlikning av komponentene

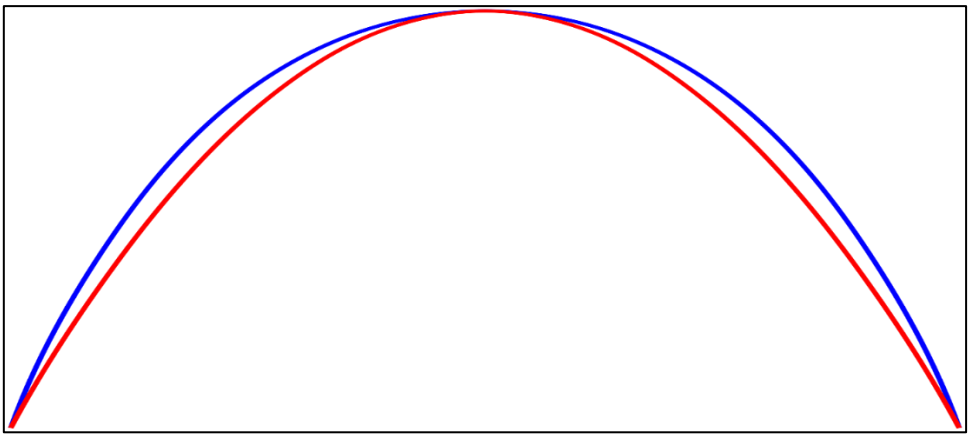
4.2.1 Sammenlikning av 2D komponenter

Alle metodene ga samme kurve dersom de ble skalert. I FDM og TNA kommer resultatet ut med en gang, mens i DR må det gjøres flere iterasjoner. Når det er gjort nok iterasjoner slik at resultatet i DR konvergerer gir den også samme resultat som FDM og TNA. Kurven som disse komponentene gir, er som forventet, en parabel. Dette er fordi lasten, som legges til i punktene, er som en jevnt fordelt last. Denne lasten er uavhengig av formen på buen, da lasten er en inndata verdi som defineres fritt. Denne verdien på lasten legges til på samme måte i alle punktene.



Figur 4-7: Sammenlikning av 2D-komponenter. TNA øverst, FDM i midten og DR nederst (med 7 iterasjoner).

Dersom lasten i stedet for regnes ut i fra hvilken form konstruksjonen har vil det gi en annen kurve. Ved å regne ut kreftene ved hjelp av densitet, gravitasjon og formen basert på tverrsnitt og lengde vil lasten oppdateres. Denne lasten brukes da på konstruksjonen, lasten oppdateres etter den nye formen, og slik forsetter det til resultatet konvergerer mot en ny form. Resultatet er en kurve som har formen til en kjedelinje. En sammenlikning av parabel og kjedelinje er vist i bildet under. Begge kurvene er laget ved hjelp av FDM, men den blå med jevnt fordelt last og den røde med last som er avhengig av formen.

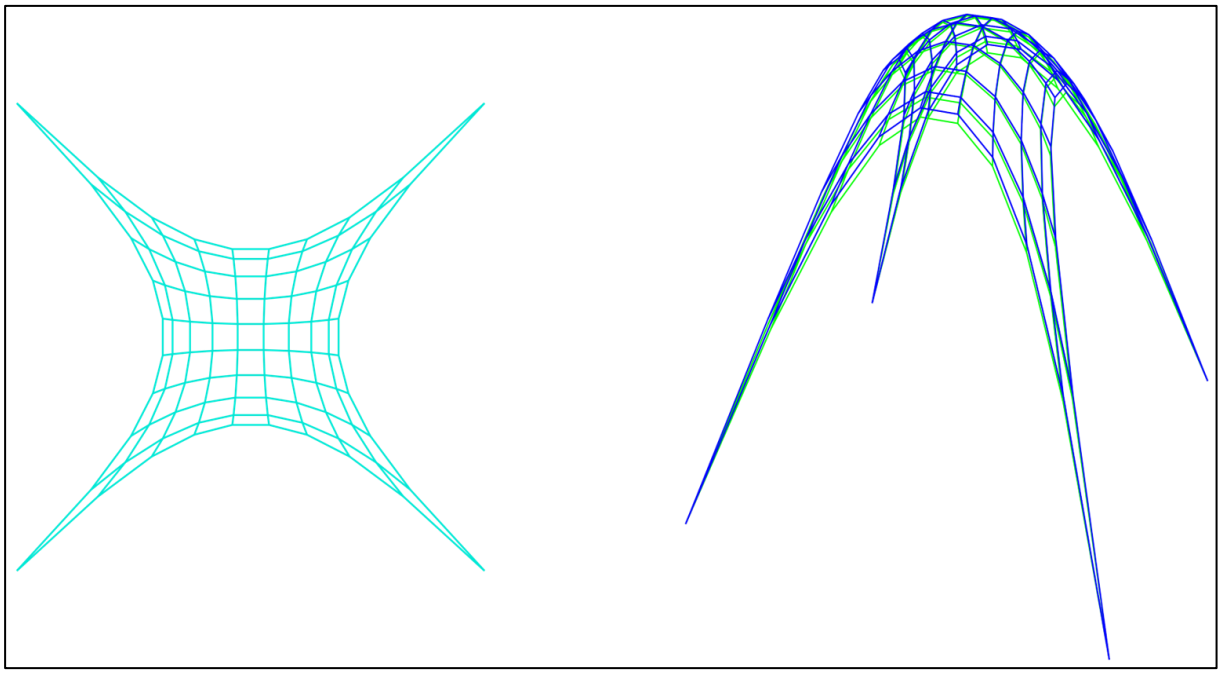


Figur 4-8: Sammenlikning av parabel (blå) og kjedelinje (rød), laget ved hjelp av FDM komponentene i Grasshopper.

Et valg man må ta hvis man skal bruke Force density method, Thrust network analysis eller Dynamic relaxation er da hvordan man skal legge på belastningen. Enten så kan man legge på en jevnt fordelt last, eller man kan ha en last som er avhengig av formen på konstruksjonen. Hvis man har en konstruksjon som skal ta opp mye ytre krefter som er jevnt fordelt i x-y planet vil det være hensiktsmessig å legge på en jevnt fordelt last. Dette er ofte tilfellet ved for eksempel hengebroer. Dersom man skal designe skallkonstruksjoner vil vanligvis den dominerende belastningen komme fra egenvekten til konstruksjonen. Det vil derfor for slike konstruksjoner være mest hensiktsmessig å la lasten være avhengig av formen på konstruksjonen.

4.2.2 Sammenlikning av 3D komponenter

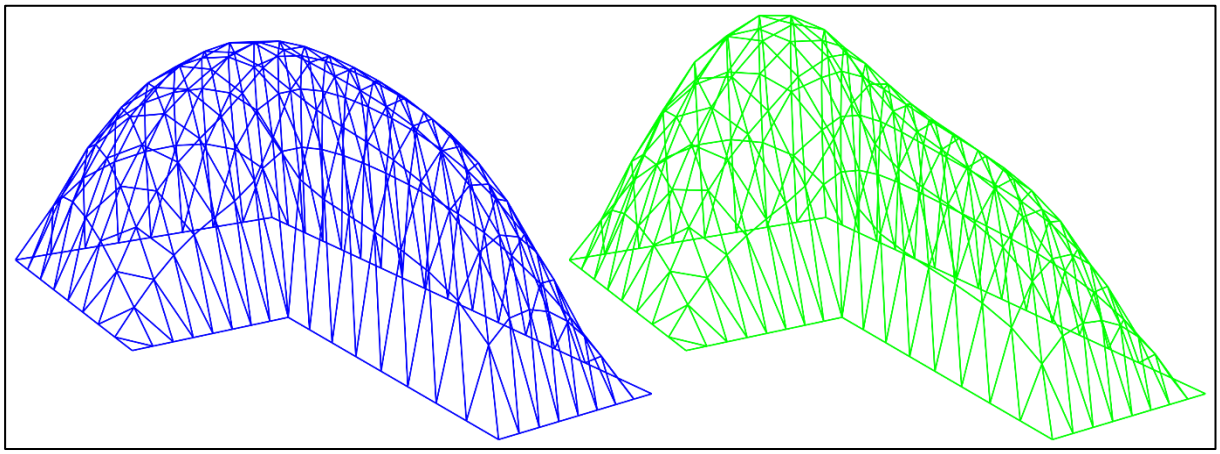
Når Force density method, Thrust network analysis og Dynamic relaxation skulle implementeres i 3D var det mange mulige måter å sette de opp på. Det ble i første omgang laget en komponent av hver metode som tok utgangspunkt i et rektangulært rutenett, med fire opplagere i hvert hjørne av rutenettet. Sammenlikning av FDM og TNA i 3D viser at de gir samme resultat med riktig skalering, akkurat som i 2D. DR gir et resultat som ser likt ut, men ved nøye sammenlikning viser det seg at DR gir en litt annen form. Den har samme koordinater i punktene i x- og y retning som FDM og TNA, men i z-retning har den litt andre verdier. I figuren under, til venstre, ser vi at alle komponentene, sett ovenfra, ser ut til å gi samme resultat. Til høyre ser vi i perspektiv at FDM og TNA (blå) gir samme resultat, mens DR (grønn) gir en litt annen form.



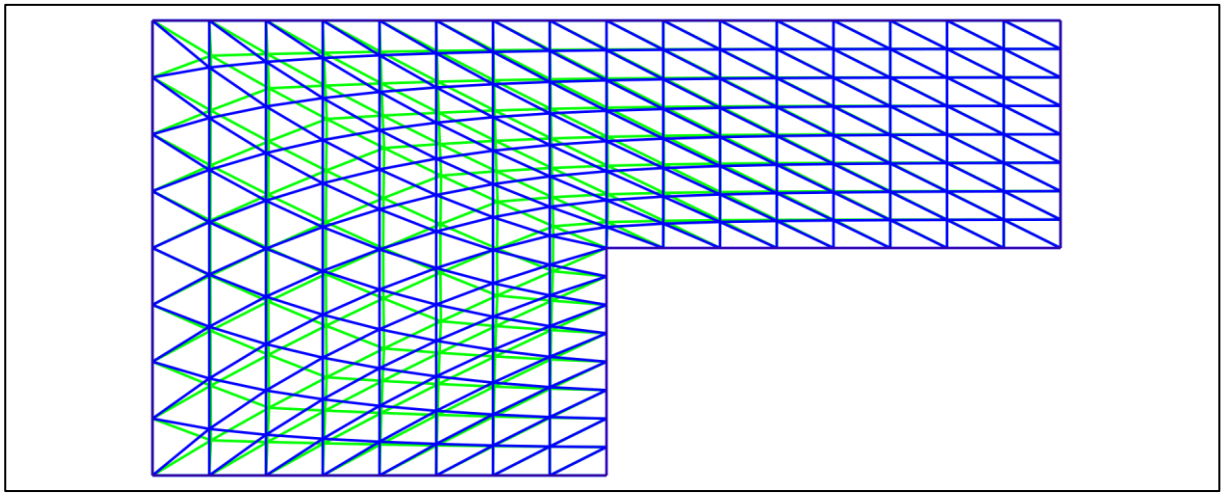
Figur 4-9: FDM (blå), TNA (blå) og DR (grønn) i 3D. Sett ovenfra (til venstre) og i perspektiv (til høyre).

Grunnen til at FDM og TNA gir samme resultat er at de settes opp veldig likt. Det eneste som skiller dem er at i TNA har krafttetthet blitt byttet ut med en skaleringsfaktor. Justerer man på skaleringsfaktoren i TNA kan man alltid kunne få den samme formen som i FDM med en viss krafttetthet, og motsatt.

DR gir derimot litt andre former enn FDM og TNA. I eksempelet over var det liten forskjell, men dersom vi tar utgangspunktet i et plan som vist i figuren under, vil DR gi en form som skiller seg enda mer fra de andre. Her varierer resultatet fra DR mye mer i høyde enn det fra FDM. FDM gir en form med mer lik høyde. I dette tilfellet er det ikke lenger bare z-koordinatene som er annerledes i DR i forhold til FDM og TNA, men også x- og y-koordinatene. Sett ovenfra vil de dermed ikke lenger se like ut.

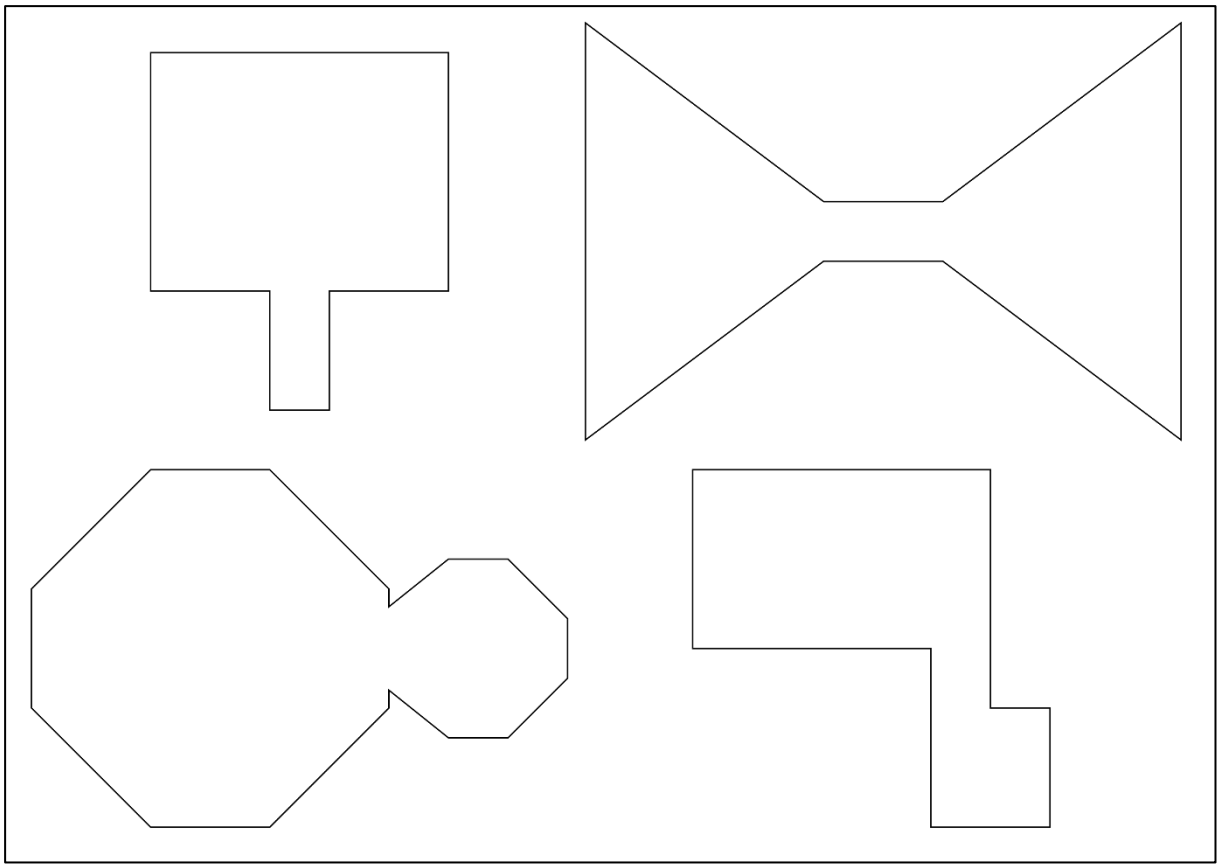


Figur 4-10: Sammenlikning av FDM (blå) og DR (grønn).

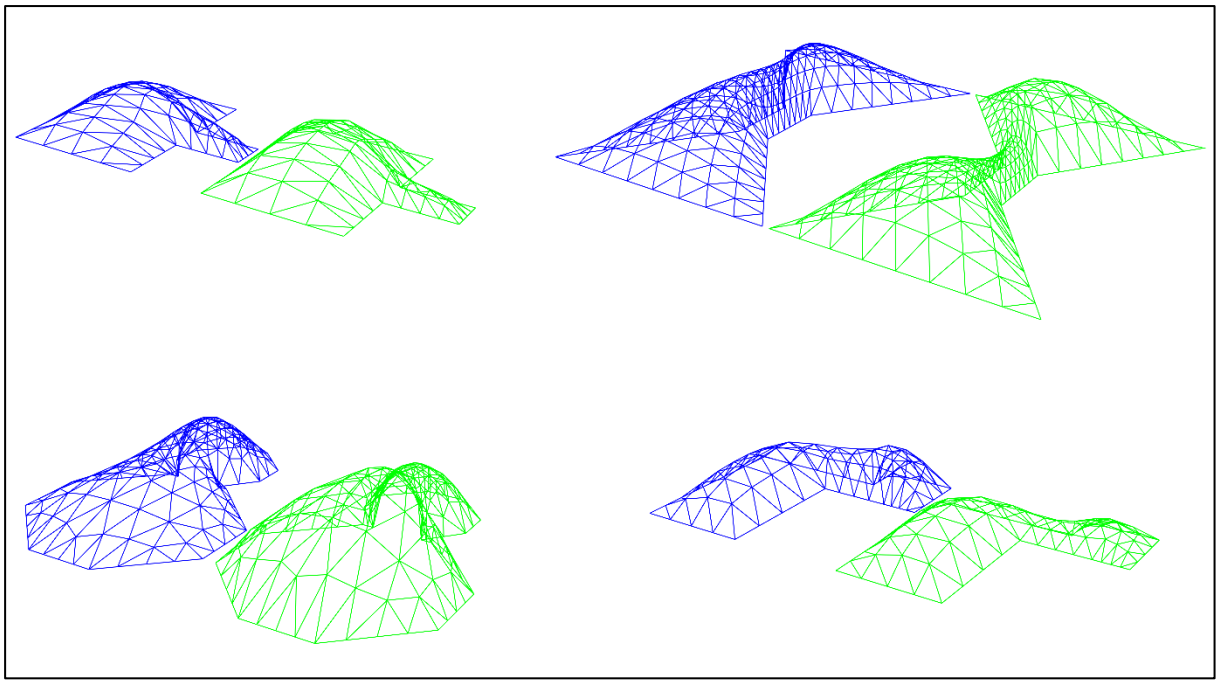


Figur 4-11: Sammenlikning av FDM (blå) og DR (grønn) sett ovenfra.

Dynamic relaxation ser ut til å gi former som er høyere der man har et stort grunnflateareal og lavere der det er mindre i forhold til Force density method. For å undersøke dette nærmere ble det laget flere modeller som ble sammenliknet. Resultatene under viser at dette er tilfellet for alle de fire forskjellige grunnflatene som ble testet med DR og FDM. Videre analyser av formene følger i neste delkapittel.



Figur 4-12: Utgangspunktet for sammenlikning av FDM og DR.



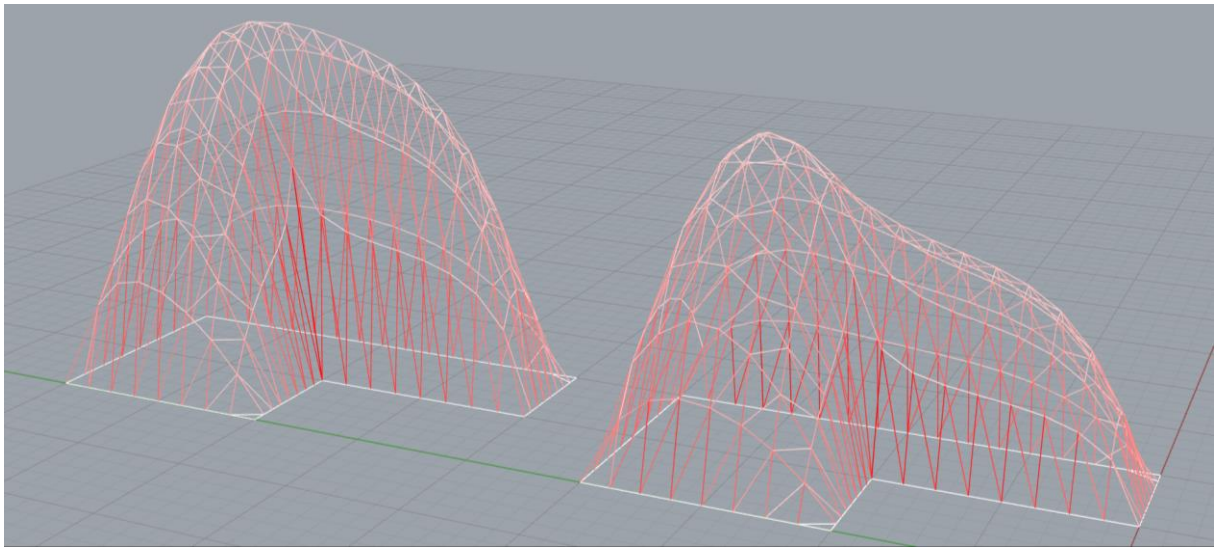
Figur 4-13: Sammenlikninger av FDM (blå) og DR (grønn). EA og indre krefter er satt lik 1.

4.2.3 Materialisering

En fordel med Force density method og Thrust network analysis er at det er færre inndata enn i Dynamic relaxation. Man får ut nye former raskere, og man slipper å ta stilling til materialegenskaper med en gang. En ulempe er at inndataene i FDM og TNA kan være noe abstrakte. I Dynamic relaxation er der mange flere inndata enn det det er i FDM og TNA. DR sine inndata er relatert til materialegenskaper og minner mer om konvensjonelle metoder. I stedet for å bruke abstrakte verdier som krafttetthet eller en skaleringsfaktor brukes materialegenskapene direkte i komponenten og resultatet vises med en gang.

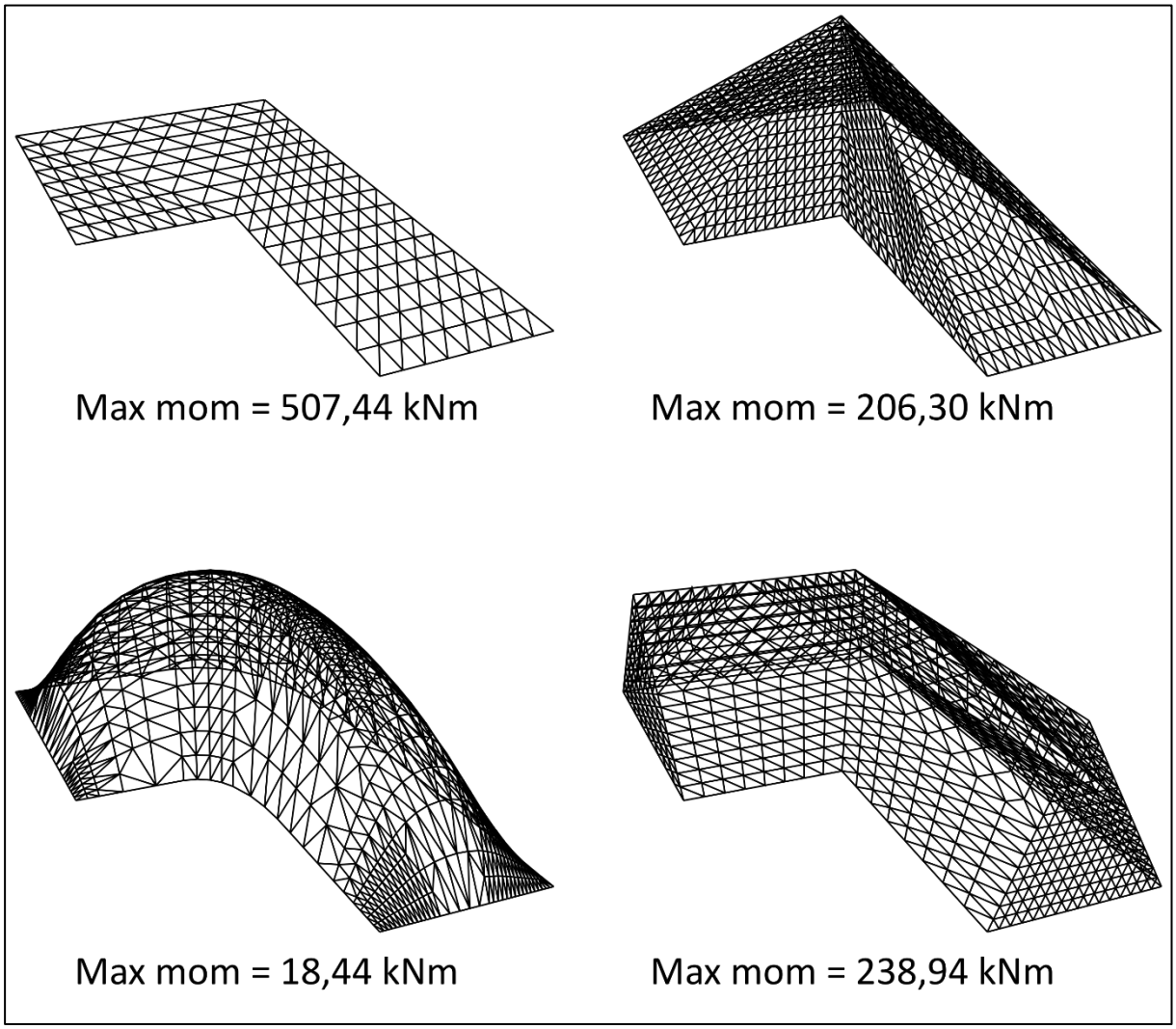
For å studere forskjellene mellom resultatene fra FDM og DR ble det kjørt en analyse av begge formene i Karamba. Analysene ble gjort ved å regne på resultatene som et gitterskall. Som utgangspunkt ble det laget en flate som var et rektangel på 20x10 m med et rektangel på 10x5 m skjært ut. Materiale som ble valgt var tre med E=1050 kN/cm2 og et kvadratisk tverrsnitt på 3x3x3 cm ble valgt. Som belastning ble det satt punktlaster i alle nodene lik 100 kN i negativ z-retning.

Variablene til FDM og DR komponentene ble justert slik at de formene som gav lavest moment ble brukt. Resultatene av analysen viser at begge formene virker hovedsakelig i trykk og momentene i konstruksjonen er små. Formen som ble funnet ved hjelp av DR gav i dette tilfellet et mindre moment enn FDM. Den største momentet-resultanten for FDM var 0,36 kNm. Dette var betydelig mer enn 0,20 kNm som var det største momentet ved DR. På figuren under ser vi hvilke bjelker som er utsatt for størst aksialbelastning. Alle bjelkene, både fra FDM og DR, som tar opp aksialkrefter virker kun i trykk.



Figur 4-14: Former funnet ved FDM (til venstre) og DR (til høyre). Større aksialkrefter vises med høyere intensitet.

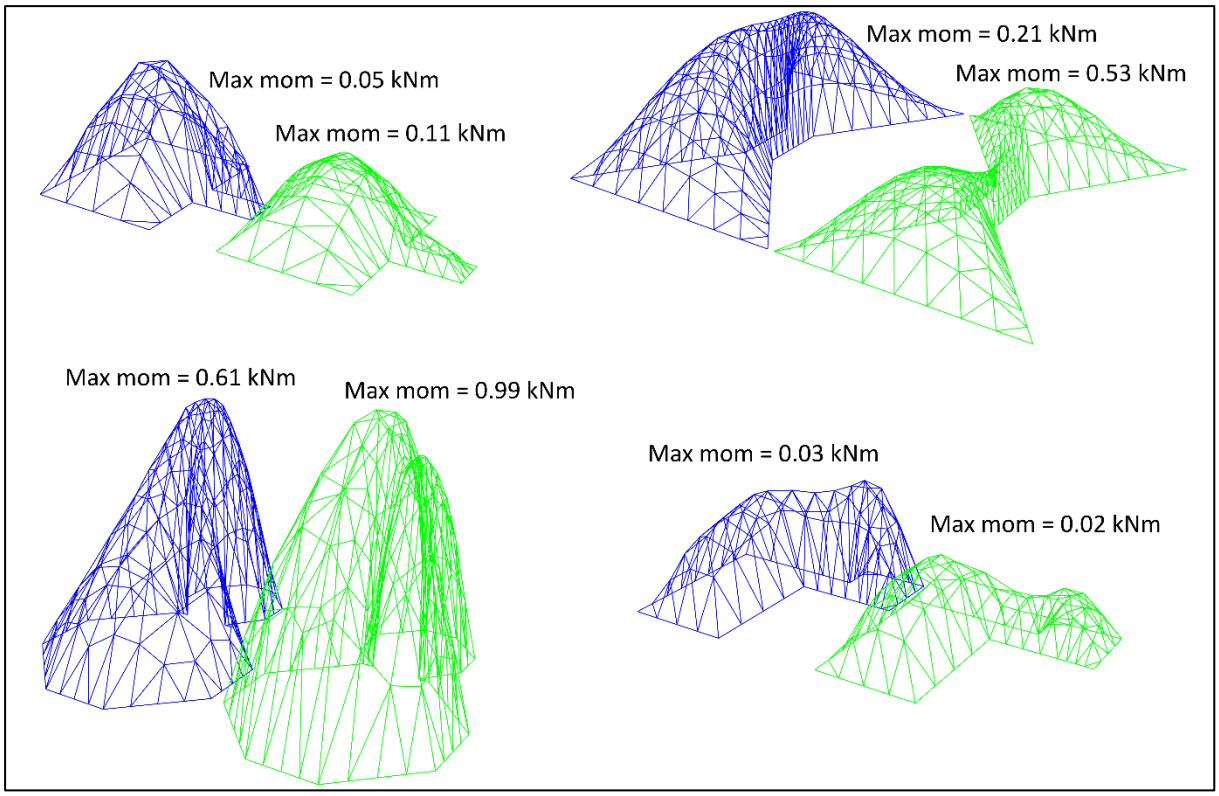
Analyser av momentene i bjelkene viser at formene som er funnet ved hjelp av FDM og DR gir bjelker med en optimalisert geometri. Momentene er små og aksialkreftene fordeler seg fint i konstruksjonene. Man kan se at aksialkreftene er små i toppen av konstruksjonen, mens de blir større nærmere opplagerne. For å undersøke hvor små momentene som ble funnet er, ble det laget flere modeller manuelt ut i fra den samme grunnflaten for å sammenlikne.



Figur 4-15: Forskjellige utforminger som ble testet for maksimal momentresultant.

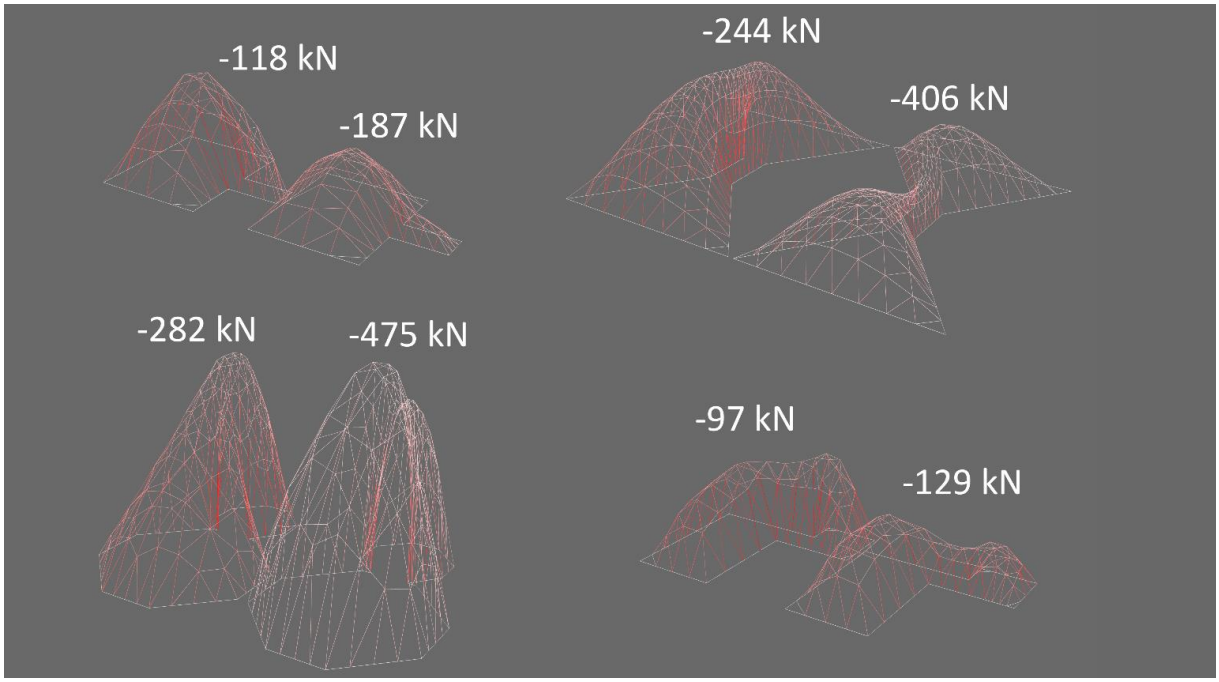
Resultatene fra disse testene viser at formene funnet ved hjelp av FDM og DR gir vesentlig mindre momenter enn disse formene. En helt flat konstruksjon sitt største moment vil være på 507,44 kNm, noe som er 2500 ganger større en momentet funnet ved DR. Selv konstruksjonen nede til venstre med moment på 18,44 kNm er nesten 100 ganger større en DR sitt på 0,20 kNm.

FDM og DR gir begge små momenter. I eksempelet over gav DR et mindre moment en FDM. For å finne ut om DR alltid gir et mindre moment en FDM ble begge metodene testet med de formene som ble definert i forrige kapittel.



Figur 4-16: Sammenlikning av momenter med FDM (blå) og DR (grønne).

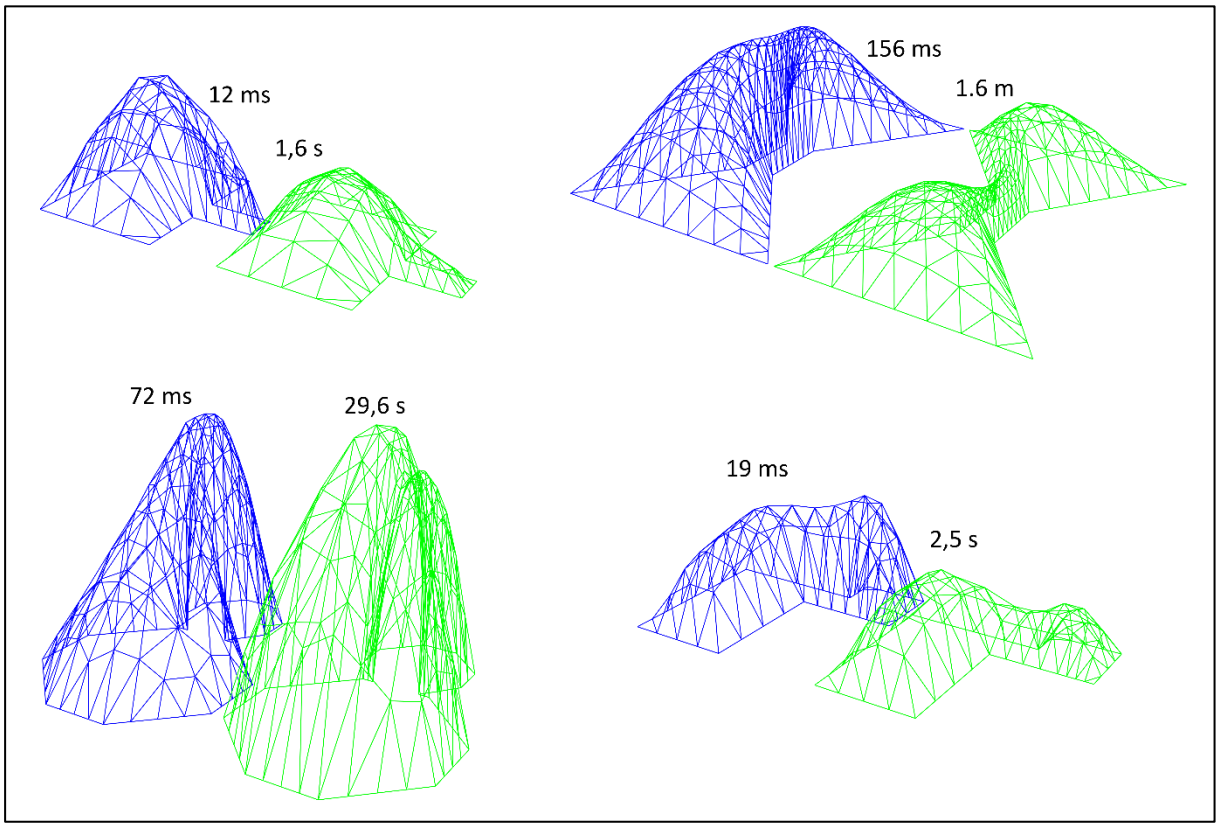
Resultatet av denne materialiseringen viser at noen ganger gir DR mindre momenter, mens i andre tilfeller gir FDM mindre momenter. De samme formene ble også sjekket med hensyn på aksialkreftene. I disse analysere, som vises på figuren under, kan man se at FDM gir minst aksialkrefter i alle de definerte tilfellene.



Figur 4-17: Sammenlikning av aksialkrefter med FDM og DR.

4.2.4 Utregningstid

Når man finner former ved hjelp av disse metodene krever de en viss utregningstid for å komme fram til likevekt. Under vises utregningstid av de samme formene som ble undersøkt for momenter og aksialkrefter.



Figur 4-18: Sammenlikning av utregningstid med FDM (blå) og DR (grønne).

DR krever flere iterasjoner for å komme frem til likevekt og bruker derfor lengre tid enn det FDM gjør på å finne likevekt. Skal man lage en komplisert geometri med mange knutepunkter og linjer vil DR bruke vesentlig lengre tid enn det FDM vil gjøre.

5 Konklusjon

Refleksjon av resultater og anbefalinger for videre arbeid.

5.1 Skallkonstruksjoner

Skallkonstruksjoner er en historisk bygningstype. Teknikken for å utforme disse konstruksjonene har utviklet seg, og bygningene har blitt mer og mer spektakulære. Force density method og Dynamic relaxation har allerede blitt benyttet i flere prosjekter på 70-tallet og i nyere tid. I de senere år har også Thrust network analysis kommet på banen og blitt anvendt.

Metodene for formfinning som er undersøkt i denne oppgaven gir konstruksjoner med veldig små momenter. Dette gjør at dimensjoneringen ikke i like stor grad som andre konstruksjonstyper fører til store tverrsnitt. Å utforme bygninger på en bedre måte gjør at man kan utnytte materialet bedre, og man får lettere konstruksjoner som følge av at man trenger mindre materiale. Man får også en bedre kraftfordeling der kraften ikke samler seg opp i noen få punkter og fører til store påkjenninger, men fordeler seg jevnt over hele konstruksjonen.

Ved å utvikle metodene som komponenter i et parametrisk dataverktøy, gjør man det mulig å teste mange mulige løsninger på kort tid. Det går raskt å utforme sofistikert geometri, komponentene kan ta utgangspunkt i en hvilken som helst form med valgfrie opplagere og med valgfritt oppsett av rutenett. Dersom man vil gjøre noen forandringer i modellen trenger man bare å endre på de relevante parameterne, og resten av modellen vil oppdateres automatisk uten å måtte bygge hele modellen opp på nytt. Dette er også en måte å designe på som gjør det enkelt å gjøre videre analyser av konstruksjonen, strukturelle eller arkitektoniske. Man kan bruke en modell i hele designprosessen uten å måtte transformere den over i andre programmer.

Hvilken av metodene som passer best for skallkonstruksjoner er ikke åpenbart. Alle metodene har fordeler og ulemper. FDM og TNA gir et raskt resultat og har få parametere. DR har materialegenskaper som komponenter, og metoden gir flere muligheter for finjustering av resultatene. Som resultatene har vist kan FDM og DR i noen tilfeller gi former som ikke er helt like. Selv om FDM i formene som ble testet ga mindre aksialkrefter enn DR, varierte det hvilken metode som ga minst momenter. Det kan da være en fordel, dersom man har mulighet, å bruke begge metodene og sammenlikne resultatene.

Metodene kan brukes i forskjellig grad. Det er helt klart ikke tilfellet at det kun er den formen som er mest optimalisert med tanke på lastbæring som er ønsket i et prosjekt. I ett bygg er det mange hensyn som skal tas og viktigheten av optimalisert geometri vil variere med ulike bygningstyper. Ofte vil det være tilfelle med større bygninger at geometrien spiller en stor rolle.

5.2 Anbefalinger

For videre arbeid kan videre materialisering av DR integreres enda bedre komponenten. Ved å bruke ekte verdier som inndata i DR er det mulig å gjøre statiske analyser direkte. Ved å legge til riktige indre krefter og aksialstivhet er det mulig at man får optimalisert formen på konstruksjonen enda litt mer. I denne oppgaven har disse verdiene for det meste bare vært konstanter som er like for alle delene i konstruksjonen.

I materialiseringsdelen av denne oppgaven ble det gjort analyser av gitterskall. Det kunne også være av interesse å se på mulighetene for å bruke disse metodene til å utforme kontinuerlige skall. Ved å lage krumme flater ut i fra linjene som komponentene finner kan det gjøres beregninger.

Det finnes flere metoder for formfinning som også kunne vært implementert. Blant annet Particle-spring system og Genetic algorithms kunne også blitt sammenliknet. Det finnes også flere utvidelser til komponentene som er undersøkt i denne oppgaven. Metodene gjøres stadig mer universelle slik at de kan brukes effektivt på flere konstruksjonstyper.

6 Referanser

[1] S. Adriaenssens, P. Block, D. Veenendaal and C. Williams, Shell Structures for Architecture: Form Finding and Optimization, Routledge, 2014.

[2] Wikipedia, «Neolittisk tid,» Wikipedia, 13 Oktober 2017. [Internett]. Available: https://no.wikipedia.org/wiki/Neolittisk_tid. [Funnet 12 Februar 2018].

[3] Wikipedia, «History of early and simple domes,» Wikipedia, 10 November 2017. [Internett]. Available: https://en.wikipedia.org/wiki/History_of_early_and_simple_domes. [Funnet 12 Februar 2018].

[4] Wikipedia, «Pantheon, Rome,» Wikipedia, 14 Februar 2018. [Internett]. Available: https://en.wikipedia.org/wiki/Pantheon,_Rome. [Funnet 19 Februar 2018].

[5] «Antoni Gaudí: Wikipedia,» Wikipedia, 9 November 2017. [Internett]. Available: https://en.wikipedia.org/wiki/Antoni_Gaud%C3%AD. [Funnet 11 November 2017].

[6] «Parametric design: Wikipedia,» Wikipedia, 23 Juni 2017. [Internett]. Available: https://en.wikipedia.org/wiki/Parametric_design. [Funnet 11 November 2017].

[7] M. Miller, «ArchDaily,» 14 April 2014. [Internett]. Available: https://www.archdaily.com/496202/ad-classics-los-manantiales-felix-candela. [Funnet 1 Mai 2018].

[8] A. Peteinarelis, «Frei Otto's contribution - Legacy to Parametric design and material computation,» Researchgate, Juli 2016. [Internett]. Available: https://www.researchgate.net/publication/305710348_FREI_OTTO'S_CONTRIBUTION_-LEGACY_TO_PARAMETRIC_DESIGN_AND_MATERIAL_COMPUTATION. [Funnet 11 November 2017].

[9] «Frei Otto: Wikipedia,» Wikipedia, 7 Oktober 2017. [Internett]. Available: https://en.wikipedia.org/wiki/Frei_Otto. [Funnet 11 November 2017].

[10] C. Meyer og M. H. Sheer, «Do Concrete Shells Deserve Another Look?,» *Concrete international,* pp. 43-50, Oktober 2005.

[11] «NAOB,» [Internett]. Available: https://www.naob.no/ordbok/parametrisk. [Funnet 6 Juni 2018].

[12] Store Norske Leksikon, «Parameter,» Store Norske Leksikon, 24 Januar 2017. [Internett]. Available: https://snl.no/parameter. [Funnet 19 Oktober 2017].

[13] Wikipedia, «Kjedelinje,» Wikipedia, 5 Oktober 2017. [Internett]. Available: https://no.wikipedia.org/wiki/Kjedelinje. [Funnet 22 Februar 2018].

[14] W. J. Lewis, Tension Structures: Form and Behaviour, London: Thomas Telford, 2003.

[15] A. Siev, «A general analysis of prestressed nets,» *Mémoires AIPC,* pp. 283-292, 1963.

[16] H.-J. Schek, «The force density method for form finding and computation of general networks,» *Computer Methods in Applied Mechanics and Engineering,* pp. 115-134, Januar 1974.

[17] P. Block, «Thrust Network Analysis,» Massachusetts Institute of Technology, Massachusetts, 2009.

[18] Wikipedia, «Dynamic relaxation,» Wikipedia, 12 Juni 2016. [Internett]. Available: https://en.wikipedia.org/wiki/Dynamic_relaxation. [Funnet 20 Februar 2018].

7 Vedlegg

Vedleggene ligger i egen zip-mappe. Her følger en oversikt over vedleggene:

Vedlegg A: Kode av FDM 2D

Vedlegg B: Kode av TNA 2D

Vedlegg C: Kode av DR 2D

Vedlegg D: Kode av FDM 3D

Vedlegg E: Kode av TNA 3D

Vedlegg F: Kode av DR 3D

Vedlegg G: Kode av FDM Flexible 3D

Vedlegg H: Kode av DR Flexible 3D