

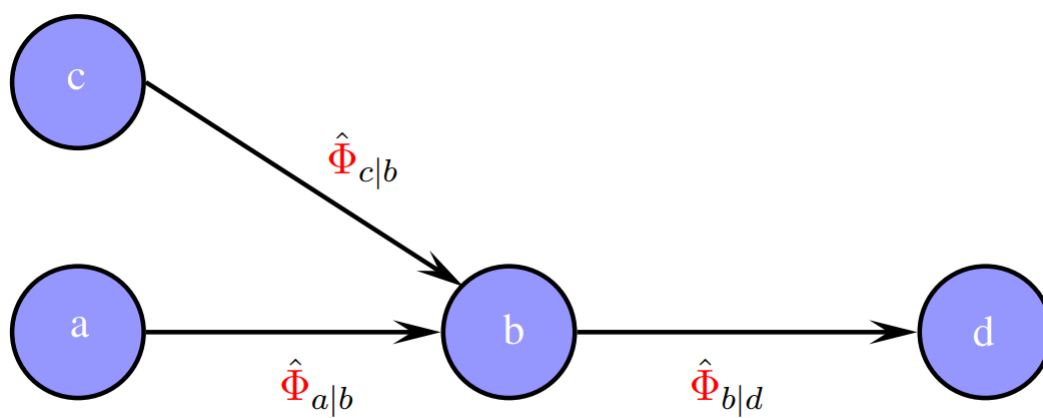
Kompendium i TKP4106 Prosessmodellering

ABC på 123

Eskild Ruud Mageli

Sindre Bakke Øyen

13. november 2018



Kompendiumskomiteen

Høiskolens Chemikerforening

styret@hc.ntnu.no

Utforming og sideombrekking: L^AT_EX

Illustrasjoner: Inkscape og paint.net

Andre illustrasjoner er printet med tillatelse fra Heinz A. Preizig.

Programmeringsverktøy: python 3.6.6

Kompendium i TKP4106 Prosessmodellering

1. utgave 2018

© Eskild Ruud Mageli og Sindre Bakke Øyen 2018

eskild.emedd33@gmail.com

sindre.bakke.oyen@ntnu.no

Høiskolens Chemikerforening har tillatelse til å distribuere bokens innhold til alle studenter på NTNU hvor innkjøpspris skal kun dekke kostnader for printing. Høiskolens Chemikerforening har ikke lov til selge boken for profitt men har lov til å runde prisen av boken opp til nærmeste ti-krone.

Medvirkende til kompendiet

Anders Runningen

Kristine Øya

Peder Langsholt Holmqvist

Joachim Ågotnes

Til min kjære 96-flaske

1 Forord

Hei og velkommen som tredjeklassing på prosess. I årene framover vil det vente deg mange morsomme, og noen krevende, emner på Institutt for kjemisk prosesssteknologi. Prosessmodellering, eller prossmod, er det første av mange fag som bygger på et fundament av modellering. Ved å mestre prossmod vil de framtidige fagene bli lettere å forstå. Dette kompendiet er en slags kortversjon av faget og har som mål å være et pedagogisk hjelpemiddel til deg som student. Forelesere på IKP har nemlig en tendens til å vinkle et ellers enkelt konsept inn i vanskelige fagtermer og mange matematiske bevis. Her vil vi prøve å bruke så mange assosiasjoner som mulig, gi eksempler som du kan kjenne deg igjen i, og legge vekt på det vi mener er det viktigste å lære seg. En foreleser vil kanskje slå hardt ned på dette og si at dette ikke er korrekt på grunn av ditten og datten, men skitt au, det rette er ikke alltid pedagogisk. Kompendiet bygger på "The ABC of Process Modelling" skrevet av Heinz Preisig. Vi presiserer at dette kompendiet ikke er offisielt pensum og gir ingen garanti for å stå i faget TKP4106 Prosessmodellering. I tillegg er det mulig faget forandrer seg med årene så kapitler i kompendiet vil ikke nødvendigvis dekke pensum fra år til år. Prossmod er et modningsfag og krever mengdetrening for forståelse, så det er viktig at du ikke tar fram dette kompendiet 3 dager før eksamen med forventning om å lære deg pensum på kort tid. Ta deg tid, gjør øvinger, diskuter faget med andre og vær kritisk til det vi skriver. Noe av det vi skriver bygger på antakelser som ikke alltid vil være realistiske, og da gjelder det å vite når man skal dykke dypere inn i emnet for en bedre forståelse. Har du spørsmål om kompendiet eller oppdager feil, så ikke nøl med å ta kontakt slik at vi kan samle opp alle endringene og eventuelt gi ut en ny utgave. Lykke til med prossmod og husk: «Hvis det går mer inn i hodet enn ut av det så akkumulerer du kunnskap».

Innhold

1	Forord	1
2	Hvordan lese dette kompendiet	4
3	Notasjon	5
4	Topologi	6
4.1	Intensive og ekstensive variabler	6
4.2	Figurer i en topologi	7
4.3	Eksempel: Et basseng når det regner	8
4.4	Oppsummert: Hvordan lage en topologi	12
5	Representere et system med ligninger	13
5.1	Balanser og konservering	13
5.2	Sette opp en enkel balanse fra en topologi	14
5.3	Eksempel: Sette opp balanser for flere systemer	14
5.4	Incidence matrix	15
5.5	Blokkmatriser	17
6	Transport og drivende kraft	18
6.1	Indre energi og effortvariabler	18
6.2	Konduksjon VS konveksjon	19
6.3	Generell transportlov	19
6.4	Lineær modell for transport	20
7	Reaksjoner og reaksjonskinetikk	22
7.1	Kjemisk potensial	23
7.2	Bestemme minimum sett med reaksjoner	23
7.3	Reaksjonsrater	25
7.4	Reaksjonsratekonstanten	26
8	Approximering av distributed systems	27
8.1	Sliced Salami method	27
8.2	Numerisk approximasjon	27
8.3	Numeriske skjemaer	29

8.3.1	Eksempel: Euler's method	29
9	Shell balance	31
9.1	Eksempel: Massestrøm gjennom en reaktor med reaksjon	31
10	Blokkdiagrammer og The Grand Scheme	34
10.1	Blokkdiagram	34
10.2	The Grand Scheme	35
11	Reduksjon av modeller med time scale og kapasitet	38
11.1	Eksempel: Tre innsjøer	38
12	Programmering	41
12.1	Løse ODE med programmering	41
12.2	Klasser og objekter	46
12.2.1	Eksempel: Katter og hunder	46
12.3	Superklasser, subklasser og arv	50
12.3.1	Eksempel: Mange forskjellige dyr	50
13	Tabell med de viktigste ligninger	52
A	Referanse	52

2 Hvordan lese dette kompendiet

Dette kompendiet er en kort oppsummering av pensum. Vi har valgt å bruke et uformelt språk for å gjøre kompendiet mer lettlest så du lettere skjønner konsepter. TKP4106 er undervist på engelsk (gitt at Heinz fortsatt underviser) så studentene vil ofte sitte igjen med engelske fagtermer. Vi har valgt å holde språket, men mikser inn engelske fagtermer der vi mener det ikke finnes en god norsk oversettelse. Faget er veldig slavisk bygd opp og følger en slags oppskrift på modellering. På samme måte har vi valgt å bygge opp kompendiet med en slags ”kokebok” på hvordan man må tenke i modellering. Vi starter på overflaten og så dykker vi dypere inn i faget for å forklare hvordan man kommer fram til forskjellige ligninger. Dette innebærer at vi bruker tidligere kapitler når vi forklarer nye konsepter, så det er anbefalt å lese kompendiet fra start til slutt når du åpner kompendiet for første gang. Til tider vil du oppdage at kompendiet har mangelfulle forklaringer. Da vil vi anbefale å bruke ABC-heftet til Heinz for mer dybde. Det er også anbefalt å friske opp i fagene Prosessteknikk og Termo GK, samt linere algebra fra Matte 3, numerikk fra Matte 4N og linearisering fra Matte 1. Til forskjell fra mange andre fag du har hatt tidligere er prossmod et fag hvor alle kapitler er koblet tett med hverandre. Det vil si: **IKKE PUGG**. Du kommer deg ikke gjennom prossmod ved å pugge deg til svarene siden det kreves forståelse for å bygge modeller. Selvfølgelig kan det hjelpe å pugge noen konsepter før man ser sammenhengen til andre kapitler. Men nøkkelordet vårt her er forståelse, og ferdigheten i å trekke røde tråder mellom de forskjellige kapitlene.

3 Notasjon

Før vi starter med å forklare konsepter tenker vi det er greit å gi en liten innføring i notasjonen som brukes i faget. Notasjonen kan forandre seg etter som faget utvikler seg, så ikke forvent at all notasjon er inkludert i dette kapittelet. Under har vi tatt en liten oppsummering av den viktigste notasjonsbruken i faget. For full forklaring av notasjon; se bakerst i ABC-heftet.

- $\hat{\underline{\mathbf{a}}}$: Strek under er en vektor
- $\underline{\underline{\mathbf{A}}}$: To streker under er en matrise
- \mathbf{A} : Tykk stor bokstav er også matrise, men sjelden brukt i dette faget.
- V : Bokstav uten strek og tykkelse er en skalar.
- n, q, m brukes henholdsvis for stoff, volum og masse
- $\dot{\underline{\mathbf{n}}}$: Dott over $\underline{\mathbf{n}}$ er akkumulert av $\underline{\mathbf{n}}$ (endring av $\underline{\mathbf{n}}$ over tid): $\frac{d(\underline{\mathbf{n}})}{dt}$
- $\hat{\underline{\mathbf{n}}}$: Hatt over $\underline{\mathbf{n}}$ transport av $\underline{\mathbf{n}}$.
- $\hat{\underline{\mathbf{n}}}_{A|B}$: Transport av $\underline{\mathbf{n}}$ fra A til B
- $\tilde{\underline{\mathbf{n}}}$: Tilde over $\underline{\mathbf{n}}$ er generert $\underline{\mathbf{n}}$ eller forbrukt $\underline{\mathbf{n}}$.
- $\underline{\underline{\mathbf{F}}}^m$: En matrise som inneholder informasjon om m, i dette tilfellet er det masse og vil representere incidence matrix
- ϕ : Phi er brukt som ekstensiv variabel, se Avsnitt 4.1
- φ : Curly phi er lik som ϕ men er en ekstensiv variabel som i tillegg avhenger av tid og retning.
- π : Ikke å forveksle med tallet 3.14. Her brukes det ofte som en effortvariabel, trykk, temperatur eller kjemisk potensial, se Avsnitt 6.1
- $:=$: kolon forran er-lik definerer en sammenheng.
- $\left. \frac{dx}{dx} \right|_i$: Stor strek ved siden siden av indikerer at uttrykket gjelder for posisjon i .
- $\left(\frac{\partial x}{\partial z} \right)_{y,w}$: Derivasjon av x m.h.p z hvor variabel y og w holdes konstant.

4 Topologi

Kort fortalt er topologi i prosessmod kunsten å dele opp et system inn i mange små kontrollvolumer som “snakker” med hverandre. En topologi inneholder ingen matematiske uttrykk, men gir en oversikt over transport av impuls, varme og/eller masse mellom forskjellige kontrollvolum. Hvordan vi setter opp en topologi er best forklart gjennom et enkelt eksempel, se Avsnitt 4.2. Men før vi går i gang må vi forklare et par konsepter.

4.1 Intensive og ekstensive variabler

Forskjellen mellom en intensiv og en ekstensiv variabel er at en intensiv variabel forblir uforandret når man skalerer opp systemet. Ta eksempelvis et badekar med vann. Vi antar at det fulle badekaret rommer 60 liter, og temperaturen er 25 grader. Hvis vi øker størrelsen på badekaret vil ikke lenger det fulle volumet (den ekstensive variabelen) være 60 liter men temperaturen (den intensive variabelen) vil fortsatt være 25 grader.

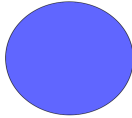


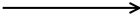



En intensiv variabel er uavhengig av størrelsen på systemet hvorav en ekstensiv variabel vil være avhengig av størrelsen.

Eksempel på intensive variabler: Temperatur, trykk, tetthet, konsentrasjon, farge, kokepunkt.

Eksempel på ekstensive variabler: Volum, masse, mol, entalpi, entropi, varmekapasitet,

4.2 Figurer i en topologi

Her kommer det en liste med forklaringer på de vanligste figurene du støter på når du skal lage en topologi i prossmod. For full liste, se ABC-heftet.

Lumped System/Kontrollvolum	
Reservoar	
Distributed system	
Transport av masse	
Transport av varme (stråling, konduksjon)	
Transport av impuls (arbeid gjort fra et system)	
Overflate uten kapasitet (event dynamic)	

Tabell 1: De viktigste geometriske figurene vi finner i en topologi. For full liste, se ABC-heftet.

Lumped System

Et lumped system representerer et volum/område for det fysiske systemet ditt. Ved å dele et stort system inn i mange små volumer kan vi få en grei forståelse av hvordan hver del av det store systemet kommuniserer. I et lumped system er det antatt at alle intensive variabler er isotrope. Det vil si at de intensive variablene er uavhengig av posisjon i systemet. Eksempelvis vil et målt trykk på 5 bar i volumet også være 5 bar alle andre steder i volumet.

Reservoar

En stor mengde som aldri går tom. Med andre ord vil et reservoar ha uendelig stor kapasitet i form av ekstensive variabler. Når det transporteres fra eller til et reservoar vil det ikke akkumuleres i reservoaret fordi reservoaret alt har uendelig mengde av den ekstensive variabelen. Alle intensive variabler i et reservoar er antatt å være

konstante så et reservoar med 25 varmegrader vil forbli 25 varmegrader.

Distributed systems

Et distributed system er som et lumped system med unntak av at intensive variabler avhenger av hvor vi befinner oss i systemet (anisotropt system). Eksempelvis vil temperaturen i en badstue være høyere jo lengre opp i høyden du beveger deg. Av den grunn kan vi ikke lenger behandle volumet av en badstue som et lumped system. Vi må sette inn et distributed system. Merk at andre intensive variabler, som trykk og tetthet, kan være antatt konstante. Et distributed system kan være anisotrop i flere dimensjoner, men må minst være avhengig av en retning. Å modellere et distributed system er litt mer avansert, se Avsnitt 8, men et lite partytriks er å bruke “Sliced-salami” metoden og dele et distributed system opp i flere mindre lumped systems.

Transportpiler

Transportpiler er transport av ekstensive variabler fra et volum til et annet. Se Avsnitt 5 for forklaring av modellering av transport. Merk at retningen på pilen er en referanse for modellen vår. Dette kan virke litt forvirrende, men det viktigste å vite er at når det transporteres fra et volum til et annet mot pilens retning, så må vi behandle transporten som negativ. Med andre ord er det ikke pilen i seg selv som bestemmer transporten, men retningen fungerer som en referanse når vi skal bestemme hvilken vei transporten går.

Overflater uten kapasitet

Egentlig burde vi ha skrevet “Event dynamic systems”, men det er litt forvirrende første gang man hører det. Det vi prøver å få fram er at vi har et system som ikke har mulighet til akkumulering siden hendelsen går så fort. Ta eksempelvis overflaten mellom vann og gass i fordampning av vann. Overflaten mellom væsken og gassen har ingen kapasitet og selve fordampningen skjer momentant. For å forklare at væske fordamper og blir til gass i topologien vår må vi ha med et event. I det tilfellet setter vi inn en rett strek, et event dynamic system.

4.3 Eksempel: Et basseng når det regner

Tenk deg at sjefen din nettopp har fått installert et nytt fancy basseng i hagen og han har satt deg i oppgave å fylle opp bassenget med vann. Du ser på værmeldingen for morgendagen og oppdager at det er meldt 15 mm i timen. «Woho», tenker du,

«bassenget fyller seg opp av seg selv». Før du begynner på regndansen funderer du på hvor mye vann som samler seg i bassenget på den tiden og du lurer på om du må fylle på med ekstra vann. Du bestemmer deg for å modellere bassenget og regnet som treffer det. Det første vi spør oss selv som ingeniører er:

1. Hva ønsker jeg å finne ut av?

I dette tilfellet er det akkumuleringen av vann i bassenget. Vi ønsker å finne ut hvor mye vann som er samlet opp i bassenget etter en viss tid. Forklart matematisk så kan vi si mengde vann i bassenget (m_{vann}) er integralet av akkumulert vann i bassenget (\dot{m}_{vann}) fra start (t_0) til slutt (t)

$$m_{vann} = \int_{t_0}^t \dot{m}_{vann} dt \quad (1)$$

Likning (1) er nå modellen vår for bassenget. Vi integrerer fra t_0 til en t også vet vi hvor mye vann som er i bassenget vårt. Men vi har et problem, vi vet ikke hvor mye vann som akkumuleres i bassenget vårt; vi vet ikke hva \dot{m}_{vann} er. Her trenger vi å sette opp en topologi. Vi må vite hvor mye vann som kommer inn og hvor mye som går ut. Før vi kan tegne opp topologien vår må vi stille oss spørsmålet:

2. Hvilke antakelser tar jeg?

Nå som vi skal visualisere systemet vårt gjennom en topologi må vi være klar over hvilke antakelser vi ønsker å gjøre i forkant av topologien. Et partytriks i prosmod er å alltid starte enkelt og så utvide topologien din ved å fjerne antakelser. I dette tilfellet vil muligens de viktigste spørsmålene være:

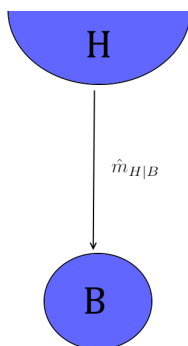
- Lekker bassenget væske?
- Er det andre kilder av vann enn regnvann?
- Skal vi behandle all væske i bassenget som et system eller skal vi skille mellom vann og annet væske?
- Vil vann fordampe og gå ut fra bassenget?
- Skal vi bry oss om varmen i bassenget?

Først når vi er klar over hva vi neglisjerer og hva som ikke kommer fram i topologien vår kan vi skrive ned antakelsene og begynne å tegne. Her velger vi først å starte

enkelt og anta at eneste kilde av væske er fra himmelen, all annen væske neglisjeres. Energibalanser ønsker vi ikke å modellere og vi antar at bassenget ikke lekker væske.

3. Tegn opp en topologi som reflekterer antakelsene

Basert på problemstillingen og antakelsene våre får vi en topologi med kun et lumped system og et reservoar. Merk at vi har valgt å se bort fra mye i starten for å få fram en enkel modell å jobbe med. Vi gjentar oss kanskje litt her, men det er fordi det er lett å forvirre seg selv. Tanken bak dette er at det er lettere å utvide en modell enn omvendt, og at det er mer strukturert å starte med det viktigste når man bygger en topologi fra scratch.



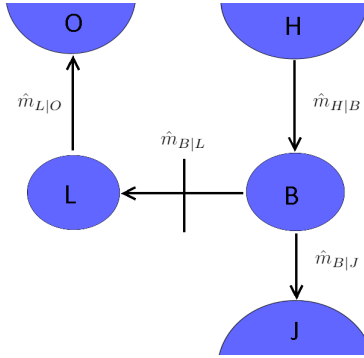
Figur 1: Topologi av regnvann inn i et basseng, H representerer himmel og B representerer basseng.

I Figur 1 ser vi representasjonen av massetransport fra H (himmel) til B (basseng). I denne topologien ser vi at all akkumulering av masse i B skyldes transporten fra H til B ($\hat{m}_{H|B}$). Nå som vi har en enkel modell kan vi vurdere hvorvidt vi skal utvide den eller ikke. Husk at en modell vil aldri være perfekt og det er bedre å ha en enkel modell som er god nok enn en vanskelig modell som knapt forbedrer.

4. Kan/bør jeg utvide topologien min?

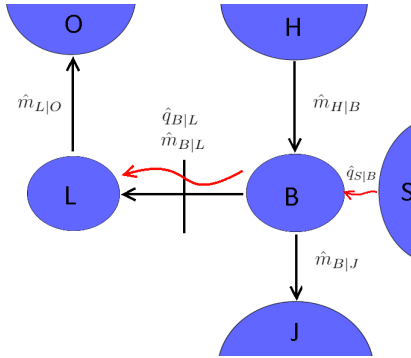
Topologien er på plass, livet er fint, men så kommer sjefen din og oppdager planen din om å la regnet gjøre arbeidet. Han fnyser av deg, men aksepterer arbeidet ditt mot at du også skal ta hensyn til lekkasje og fordampingen av vann i bassenget. “Fillern” tenker du og hopper tilbake til arbeidet. Når vi utvider en modell er det viktig å gå tilbake på antakelsene og revurdere dem før vi utvider. For å få fram fordampningen og for å vise at bassenget vårt består av et volum av væske og et volum av gass setter vi opp to lumped systems med et event dynamic system mellom seg (overflaten). I tillegg antar vi at lekkasjen av væske skjer til bakken og at luften

i bassenget forsvinner ut til omgivelsene.



Figur 2: Topologi av et basseng. H er en regnfull himmel, B er vannet i bassenget, L er luften i bassenget, O er luften i omgivelsene rundt bassenget, J er bakken.

Den nye topologien er på plass og livet er nok en gang fint. Vi har laget en topologi hvor vi kan lage balanser rundt to volumer. Sjefen vår er ikke helt fornøyd og forventer at vi samtidig må ta hensyn til varmen i modellen vår. Vi sukker dypt, men etter litt tankegang kommer vi fram til en endelige topologi:



Figur 3: Topologi av et basseng. H er en regnfull himmel, B er vannet i bassenget, L er luften i bassenget, O er luften i omgivelsene rundt bassenget, J er bakken, S er solen.

I denne topologien har vi antatt at solen er det eneste reservoaret av varme. Vi har neglisjert oppvarming av luften fra sola. Varmetransporten mellom B og L skyldes at hvis B skal fordampe kreves det energi i form av fordampningsentalpi. varmetransporten fra bassenget til jorda neglisjeres i denne modellen. Dette kan argumenteres med at temperaturforskjellen mellom bassenget og jorden er minimal så tapet av energi er ikke betydelig til modellen vår. Det er ganske forvirrende med modellering av varmetransport og du er ikke den eneste på IKP som synes det er vanskelig. Likevel er det en grei tommelfingerregel når det kommer til modellering a varmetransporten. Massetransport er all transport som følge av konveksjon mens varmetransport er all

transport som følge av stråling og konduksjon, mer om dette i Avsnitt **6.2**. Heldigvis er vi nå ferdig med topologien vår. Vi leverer arbeidet vårt til sjefen som sier seg fornøyd og spanderer en Big Mac på oss.

5. Ingen topologier er perfekte

Nå som vi har laget en topologi og ønsker å bruke den til å lage en matematisk modell for volumet i bassenget er det viktig å være klar over begrensninger ved modellen vår. For eksempel er det godt mulig at antakelsen vår om at isotropisk oppførsel for temperaturen i bassenget og luften ikke er en god antakelse. En mulighet kunne vært å sette inn et distributed system istedet, men det ville samtidig komplisere modellen vår. Ved å være klar over hva topologien ikke framstiller er vi også klar over hvilke begrensninger modellen vår har når vi bruker den. Å vite hva som er svakhetene ved en modell er essensielt siden du ikke har LF ute i arbeidslivet.

4.4 Oppsummert: Hvordan lage en topologi

1. Finn ut hva du ønsker å finne ut av. Er det modellering av masse, energi, impuls eller en kombinasjon?
2. Hvilke antakelser er fornuftige? Start med det essensielle for problemet
3. Tegn opp en topologi som reflekterer antakelsene
4. Utvid topologien og modellen til du har en «god nok» representasjon av det fysiske systemet ditt

5 Representere et system med ligninger

Når vi snakker om å representere et system med ligninger mener vi å sette opp balanseligninger. I prossmod er det et fåtall av ligninger du trenger å kunne, men en av de viktigste er hvordan vi setter opp en balanse for et lumped system/kontrollvolum.

$$\text{Akkumulering} = \text{Inn} - \text{Ut} + \text{Generert} \quad (2)$$

Kort fortalt ønsker vi å sette opp en ligning for hvordan en ekstensiv variabel endrer seg inne i volumet. Fra Likning (2) ser vi at endringen inne i volumet er avhengig av transporten inn og ut, og hvor mye som genereres/forbrukes inni volumet. Merk at vi bare kan sette opp en slik balanse for ekstensive variabler siden intensive variabler er uavhengig av størrelsen på systemet. Vi bruker variabelen Φ som en vilkårlig ekstensiv variabel videre i kapittelet.

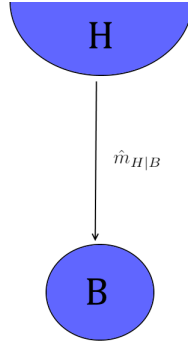
5.1 Balanser og konservering

$$\text{Konservering: } \dot{\Phi}_{system} = \hat{\Phi}_{inn} - \hat{\Phi}_{ut} \quad (3)$$

$$\text{Balanse: } \dot{\Phi}_{system} = \hat{\Phi}_{inn} - \hat{\Phi}_{ut} + \tilde{\Phi}_{system} \quad (4)$$

En konservering er en balanse hvor genereringen av den ekstensive variabelen er null, dvs. all akkumulering skyldes transport inn og ut av systemet. Et system kan ofte bestå av balanser og konserveringer. For eksempel i en reaktor vil det genereres stoffmengde, men den totale massen vil være konserverert. Du kommer deg fint gjennom livet ved bare å bruke terminologien balanser, men vær klar over at en balanse blir en konservering når det ikke er generering/forbruk, det vil si $\tilde{\Phi}_{system} = 0$. For å ikke forvirre leseren vil vi bruke terminologien balanse videre i kompendiet.

5.2 Sette opp en enkel balanse fra en topologi



Figur 4: Topologi av regnvann inn i et basseng, H representere himmel og B representerer bassenget.

Det er kun rundt systemer med kapasitet vi kan gjøre balanser (lumped/distributed). I vår topologi, gitt i Figur 4, er H et reservoar og er ikke interessant å gjøre en balanse rundt siden kapasiteten er uendelig ($\dot{\Phi}_H = 0$). Volumet B kan vi sette opp en balanse rundt ved å bruke Likning (4):

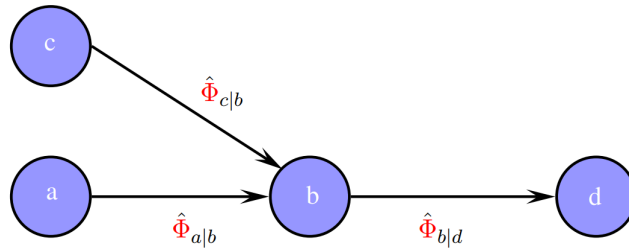
$$\dot{\Phi}_B = \hat{\Phi}_{H|B} \quad (5)$$

Vi vet alt at vår ekstensive variabel fra topologien er masse så kan vi bytte ut Φ med m :

$$\dot{m}_B = \hat{m}_{H|B} \quad (6)$$

Vi har nå en ligning for systemet vårt og vi kan regne ut akkumuleringen av masse i B hvis vi vet transporten fra H til B.

5.3 Eksempel: Sette opp balanser for flere systemer



Figur 5: Topologi av fire lumped systems med transport av en ekstensiv variabel. Figur hentet fra ABC-heftet.

Vi har et system bestående av fire lumped systems. Ved å benytte oss av samme teknikk som i kapittel 5.2 setter vi opp fire balanser:

$$\begin{aligned}\dot{\Phi}_a &= -\hat{\Phi}_{a|b} \\ \dot{\Phi}_b &= \hat{\Phi}_{a|b} + \hat{\Phi}_{c|b} - \hat{\Phi}_{b|d} \\ \dot{\Phi}_c &= -\hat{\Phi}_{c|b} \\ \dot{\Phi}_d &= \hat{\Phi}_{b|d}\end{aligned}\tag{7}$$

5.4 Incidence matrix

I modellering får vi ofte et system som består av flere titalls mindre systemer som gjør det vanskelig å holde styr på alle balansene. Det er derfor ønskelig å benytte seg av vektorer og matriser.

Ved utgangspunkt i (likningssett) 7 definerer vi vektoren for akkumulering og transport:

$$\begin{aligned}\text{Vektor for akkumulering: } \underline{\dot{\Phi}} &= [\dot{\Phi}_a, \dot{\Phi}_b, \dot{\Phi}_c, \dot{\Phi}_d] \\ \text{Vektor for transport: } \underline{\hat{\Phi}} &= [\hat{\Phi}_{a|b}, \hat{\Phi}_{c|b}, \hat{\Phi}_{b|d}]\end{aligned}$$

Vi bruker disse vektorene til å representere balansene fra (likningssett) 7 i en samlet matrise som har navn *Incidence Matrix*.

$$\underline{\underline{\mathbf{F}}} = \begin{matrix} & \hat{\Phi}_{a|b} & \hat{\Phi}_{c|b} & \hat{\Phi}_{b|d} \\ \begin{matrix} \dot{\Phi}_a \\ \dot{\Phi}_b \\ \dot{\Phi}_c \\ \dot{\Phi}_d \end{matrix} & \begin{pmatrix} -1 & 0 & 0 \\ +1 & +1 & -1 \\ 0 & -1 & 0 \\ 0 & 0 & +1 \end{pmatrix} \end{matrix}\tag{8}$$

Hver rad i incidence matrix representerer en balanse og hver kolonne representerer transport fra et volum til et annet. Ved å sette opp incidence matrix kan vi skrive likningsettet 7 som:

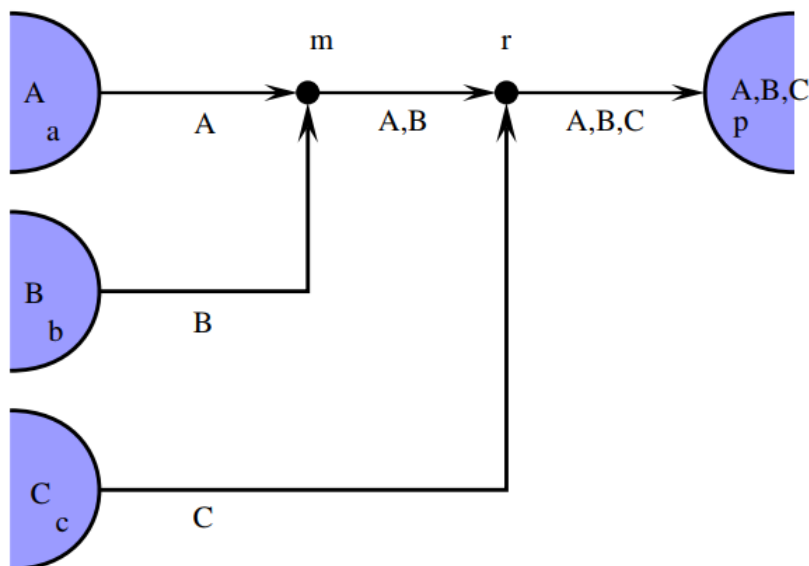
$$\underline{\dot{\Phi}} = \underline{\underline{\mathbf{F}}} \underline{\hat{\Phi}}\tag{9}$$

Her har vi nå satt sammen et likningsett inn til en ligning med vektorer og matriser. Senere i kompendiet vil vi fortsette med denne notasjonen for å vise den lineære

algebraen som kommer fram når vi modellerer. Bakgrunnen for å bruke denne notasjonen fremfor å forklare det med likninger er fordi lineær algebra har strengere krav for operasjoner enn operasjoner på skalarer i en enkel likning. Hvis du nå tenker “Men jeg husker ikke så mye fra Matte 3”, sorry brah! Lineær algebra kommer du deg ikke utenom på IKP. Likevel, hvis du føler det er vanskelig å forholde seg til notasjonen så kan det hjelpe å skrive ut matrisen til et sett med ligninger og forholde seg til dem istedet.

5.5 Blokkmatriser

Når man skal sette opp en incidence matrix for flere komponenter, e.g. stoffmengder, kan det være lurt å bruke matrisen til å sette opp en blokkmatrise. Dette er vanlig når man har flere komponenter som beveger seg mellom forskjellige volum.



Figur 6: Topologi av system m og r som transporerer stoffene A,B og C. Figur hentet fra ABC-heftet.

flows →		$a m$	$b m$	$c r$	$m r$		$r p$		
↓ system ↓		A	B	C	A	B	A	B	C
m	A	1			-1				
m	B		1			-1			
r	A				1		-1		
r	B					1		-1	
r	C			1					-1

Figur 7: Incidence block matrix for topologien presentert i Figur 6. Figur hentet fra ABC-heftet.

Vi dekker ikke denne delen i kompendiet siden prinsippet er mye av det samme, men oppfordrer sterkt å lese kapittel 7.1 i ABC-heftet hvis du ikke forstår deg på blokkmatriser.

6 Transport og drivende kraft

Til nå har vi antatt at vi kjenner transporten mellom hvert system. Hva gjør vi hvis vi ikke kjenner denne transporten? Svaret er kanskje ikke overraskende; Vi modellerer den! Men hvordan modellerer vi transporten ($\hat{\Phi}_{a|c}$, $\hat{\Phi}_{b|c}$, ...) mellom de forskjellige systemene? Spenn setebeltet fast for nå beveger vi oss inn i termodynamikken.

6.1 Indre energi og effortvariabler

Endringen av en ekstensiv variabel $\hat{\Phi}$ er bestemt av intensive variabler. Vi gidder ikke bruke mye tid på å forklare all matematikken bak drivende kraft (Kapittel 5.1 i ABC-heftet) så vi hopper rett inn i det som kalles *effortvariabler* (π). Effortvariabler er intensive variabler definert som endringen av indre energi U ved forskjellige kriterier. Indre energi er en funksjon av entropi, volum og stoffmengde (S, V, \underline{n}) som vi finner i ligningen for total energi. I ligningene under ser vi sammenhengen mellom indre energi og effortvariabler. Notasjonen med underindekser til brøken betyr at de variablene holdes konstant.

$$\begin{aligned}\left(\frac{\partial U}{\partial S}\right)_{V, \underline{n}} &=: T \\ \left(\frac{\partial U}{\partial V}\right)_{S, \underline{n}} &=: -p \\ \left(\frac{\partial U}{\partial \underline{n}}\right)_{S, V} &=: \mu\end{aligned}\tag{10}$$

Det viktigste å huske fra Ligningene i 10 er at temperatur, trykk og kjemisk potensial er effortvariablene som sørger for endringer i ekstensive variabler. Med andre ord er det endring i T , p og μ som sørger for transport fra et system til et annet. For eksempel vil varmetap i en bolig skje fordi temperaturen utenfor er lavere enn temperaturen inne. Det vil si at forskjellen mellom temperaturen ute og inne sørger for transport av varme ut fra boligen.

Ekstra:

Konsentrasjon og molbrøk er ikke effortvariabler, men istedet utledet fra kjemisk potensial. Ofte brukes konsentrasjon i uttrykk for transporten i form av diffusjon eller kjemisk reaksjon. Det er ikke nødvendigvis feil å bruke konsentrasjon i beregningene, men ved likevekt vil ikke nødvendigvis konsentrasjonsforskjellen mellom de

to stoffmengdene være null, men forskjellen i det kjemiske potensialet vil alltid være null.

6.2 Konduksjon VS konveksjon

Dette er to ord som ofte blir brukt om hverandre i varmetransport og som kan forvirre mange.

- **Konduksjon:** overføring av varme fra et molekyl til et annet via gnisninger mellom molekyler
- **Konveksjon:** varmeoverføring på grunn av transport av masse med høy indre energi

Eksempel på konveksjon er kulden du merker når du åpner et vindu på en kald vinterdag. Alle de kalde luftmolekylene, med lavere indre energi enn molekylene i rommet, transporteres inn i stuen din og kjøler deg ned. Konduksjon er transporten av varmen gjennom veggen og ut til den kalde luften. Hvorfor er dette viktig å vite, spør du. Jo, for når du skal tegne opp en topologi for varmetransport vil ikke konveksjon behandles likt som konduksjon i en modell. Konveksjon behandles som transport av indre energi (som oftest entalpi). Men ikke tenk for mye på dette. PS: Stråling er også en form for varmetransport som du sikker husker fra Strømningsfaget.

6.3 Generell transportlov

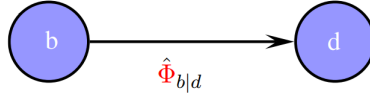
Hvis vi har transport av en ekstensiv variabel fra et system til et annet hvor posisjonen i system er betydelig for transporten ($\hat{\varphi}$), så er det vanlig å sette opp en differensialligning for endringen av effortvariabelen. NB: her bruker vi $\hat{\varphi}$ istedet for $\hat{\Phi}$ siden den ekstensive variabelen er avhengig av posisjon.

$$\hat{\varphi} = -c \frac{\partial \pi}{\partial \underline{\mathbf{r}}} \quad (11)$$

c er konduktiviteten (uttrykk for motstanden) og $\underline{\mathbf{r}}$ er retningene som $\hat{\varphi}$ transporteres gjennom. Denne likningen ligner på Ficks lov eller Fouriers lov som du, forhåpentligvis, har sett tidligere i henholdsvis RekTek og Termo GK. Til vår glede er Likning (11) Ficks og Fouriers lov på generell form.

Eksempel:

Si at vi har en topologi som er representert med to lumped systems med transport mellom de to systemene, se Figur 8.



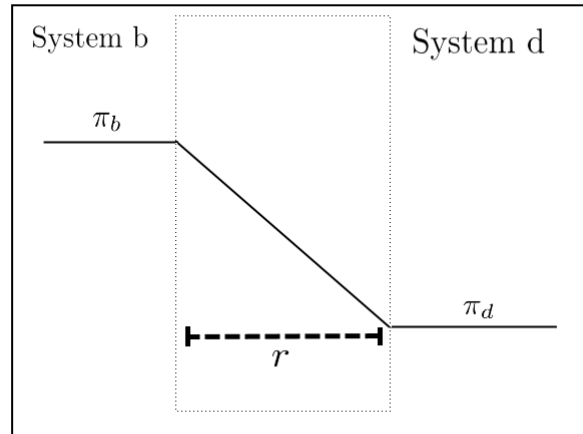
Figur 8: Topologi av to lumped systems med transport fra b til d .

Bruker vi Likning (11) for transporten mellom b og d ser vi at transporten, $\hat{\Phi}_{b|d}$, er avhengig av endringen av π i system b og d :

$$\hat{\Phi}_{b|d} = \hat{\varphi}_{b|d} = -c_{b|d} \frac{\partial (\pi_{b|d})}{\partial r_{b|d}} \quad (12)$$

6.4 Lineær modell for transport

Anta at vi er interessert i en enkel modell for transporten mellom b og d , som vist i Figur 8, og ikke har informasjon om hvordan π endrer seg fra b til d . Da er det rimelig å benytte seg av en lineær modell for transporten mellom systemene. Vi lineariserer endringen av effortvariablen fra system b til d og får en modell som er grafisk representert i Figur 9.



Figur 9: Grafisk representasjon av lineariseringen av $\frac{\partial \pi}{\partial \mathbf{r}}$ som viser den lineære endringen i effort-variabelene fra system b til system d . r er avstanden mellom systemene.

Lineariseringen av $\frac{\partial \pi}{\partial \mathbf{r}}$ gjøres ved bruk av en Taylor utvidelse av første orden, se Avsnitt 8. Dette gir oss en lineær modell for transporten av den ekstensive variabelen:

$$\hat{\Phi}_{b|d} = -k_{b|d}(\pi_b - \pi_d) \quad (13)$$

Denne likningen kan vi nå substituere inn i balansene fra Avsnitt 5. Merk at vi har en lineær modell som ikke nødvendigvis er korrekt. Hvis transporten mellom to systemer er essensielt for modellen kan det være lurt å benytte seg av andre metoder for modellering av transport f. eks. Likning (11), Governing equations (ikke pensum), Navier Stokes (ikke pensum) eller shell balance.

7 Reaksjoner og reaksjonskinetikk

Reaksjoner sies å være fundamentet i kjemisk industri. Reaksjoner er interaksjoner mellom stoffer som skaper nye stoffer. I Avsnitt **5.2** så vi hvordan vi kan sette opp en balanse med Likning (4). I dette kapitlet er vi bare interessert i stoffmengde så vi bytter ut Φ med n . Denne delen viser fram mange likninger så det er bare å holde tunga rett i munnen. Vi skal prøve å gå gjennom dem en etter en. Vi starter med å sette opp den generelle balansen for stoffmengde i systemet:

$$\dot{\underline{n}} = \hat{\underline{n}} + \tilde{\underline{n}} \quad (14)$$

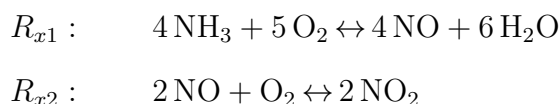
La oss si at systemet vårt ikke har transport inn og ut (e.g. batch reaktor, $\hat{\underline{n}} = 0$) så all akkumulering i systemet vårt skyldes reaksjoner innad i systemet vårt. Uten å bruke mye tid på forklaring så er genereringsleddet i balansen definert som:

$$\begin{aligned} \dot{\underline{n}} &= \tilde{\underline{n}} \\ \tilde{\underline{n}} &= V \underline{\underline{\mathbf{N}}}^T \tilde{\underline{\xi}}(\underline{\mathbf{c}}) \end{aligned} \quad (15)$$

$\underline{\underline{\mathbf{N}}}^T$ er den støkiometriske matrisen til reaksjonene, $\tilde{\underline{\xi}}$ er reaksjonsraten som en funksjon av konsentrasjoner og V er volum. Volum ganges inn i likningen for at $\tilde{\underline{n}}$ skal ha riktig benevning. Reaksjonsraten er en funksjon av konsentrasjon og reaksjonsratekonstant.

Eksempel: Stoichimetric species matrix

Den støkiometriske matrisen blir laget ved å sette alle reaksjonene på hver kolonne og hver reaktant/produkt på hver rad.



$$\underline{\underline{\mathbf{N}}} = \begin{matrix} & R_{x1} & R_{x2} \\ \text{NH}_3 & \left(\begin{array}{cc} 4 & 0 \\ 5 & 1 \\ 4 & 2 \\ 6 & 0 \\ 0 & 2 \end{array} \right) \\ \text{O}_2 & \\ \text{NO} & \\ \text{H}_2\text{O} & \\ \text{NO}_2 & \end{matrix} \quad (16)$$

Husk at vi jobber med den transponerte av $\underline{\underline{\mathbf{N}}}$ i Likning (15).

7.1 Kjemisk potensial

Men hvordan vet vi hvilken vei som vil dominere reaksjonen? I Avsnitt 6 lærte vi om effortvariabler som driver transport. I kjemiske reaksjoner er det kjemisk potensial som bestemmer likevekten mellom reaktantene og produktene:

$$\mu_i = \left(\frac{\partial U}{\partial n_i} \right)_{S,V,n_{j \neq i}} \quad (17)$$

Likevel er det vanlig å bruke konsentrasjon som variabel i reaksjonlikninger siden konsentrasjon er lettere å fysisk måle enn kjemisk potensial. Det viktig å vite at med kjemisk potensial i dette avsnittet er at den favoriserte reaksjonen er den som skaper den laveste endringen i kjemisk potensial $\Delta\mu$.

En alternativ måte å representere kjemisk potensial er i form av avvik fra ideell løsning. Istedet for å gå gjennom utledning fra energibalanser er denne metoden empirisk og tar utgangspunkt i kjent data. Med andre ord så regnes det kjemisk potensialet fra residualet til den ideelle løsningen.

$$\begin{aligned} \mu_i &= \mu_i^0 + \mu_i^{excess} \\ \mu_i &= \mu_i^0 + RT \ln(x_i) \end{aligned} \quad (18)$$

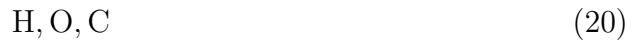
7.2 Bestemme minimum sett med reaksjoner

La oss si at vi ikke vet reaksjonene, men har fått vite at vi har en mengde forskjellige stoffer og ønsker å finne ut hvilke reaksjoner som skjer.

Eksempel: Vi har reaktantene:



Ved å se på stoffene i Likning (19) trekker vi fram hvilke atomer de 5 stoffene er bygd opp av. Her har vi 3 forskjellige atomer.



Ved bruk av Likning (19) og Likning (20) konstruerer vi en matrise som viser hvor mange atomer hvert stoff består av.

$$\underline{\underline{\mathbf{C}}} = \begin{matrix} & \text{H}_2\text{O} & \text{CH}_4 & \text{CO} & \text{CO}_2 & \text{H}_2 \\ \begin{matrix} \text{H} \\ \text{O} \\ \text{C} \end{matrix} & \begin{pmatrix} 2 & 4 & 0 & 0 & 2 \\ 1 & 0 & 1 & 2 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \end{matrix} \quad (21)$$

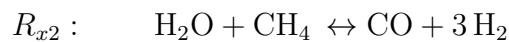
For å finne antall reaksjoner må vi finne nullspacet til matrisen ($\text{null}(\underline{\underline{\mathbf{C}}})$). Vi ønsker å finne ut hvor mange av radene i $\underline{\underline{\mathbf{C}}}$ som er lineært avhengige (antall kolonner minus ranken til $\underline{\underline{\mathbf{C}}}$). Metodikken er å sette matrisen lik null og radredusere (Gauss reduction, løse matrisen) slik vi gjorde i Matte 3. Fordi vi er late og regner med at din lineære algebra er på topp gidder vi ikke å vise alle operasjonene (se ABC-heftet kapittel 8.2 hvis du er usikker):

$$\underline{\underline{\mathbf{C}}} = \begin{bmatrix} 3 & 0 & 0 & 2 & 1 \\ 0 & 3 & 0 & -1 & 1 \\ 0 & 0 & 3 & 4 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 2/3 & 1/3 \\ 0 & 1 & 0 & -1/3 & 1/3 \\ 0 & 0 & 1 & 4/3 & -1/3 \end{bmatrix} \quad (22)$$

$$\text{null}(\underline{\underline{\mathbf{C}}}) = \begin{bmatrix} -2 & -1 \\ 1 & -1 \\ -4 & 1 \\ 3 & 0 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} -2/3 & -1/3 \\ 1/3 & -1/3 \\ -4/3 & 1/3 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (23)$$

Fra nullspacet vårt ser vi at $\underline{\underline{\mathbf{C}}}$ har 2 uavhengige rader som betyr at $\underline{\underline{\mathbf{C}}}$ har 2 frie variabler. Med andre ord så er ranken til $\underline{\underline{\mathbf{C}}} = 3$. Nå som vi vet at vi har to frie variabler kan vi konstruere 2 reaksjoner fra nullspacet vårt:

$$\underline{\underline{\mathbf{N}}} = \begin{array}{c} \text{H}_2\text{O} \\ \text{CH}_4 \\ \text{CO} \\ \text{CO}_2 \\ \text{H}_2 \end{array} \begin{array}{cc} R_{x1} & R_{x2} \\ \left(\begin{array}{cc} -2 & -1 \\ 1 & -1 \\ -4 & 1 \\ 3 & 0 \\ 0 & 3 \end{array} \right) \end{array} \quad (24)$$



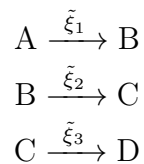
7.3 Reaksjonsrater

$$\tilde{\xi}(\underline{\mathbf{c}}) := \underline{\underline{\mathbf{K}}} g_r(\underline{\mathbf{c}}) \quad (25)$$

$\underline{\underline{\mathbf{K}}}$ er en diagonal matrise bestående av reaksjonratene $(\xi)^1$, hvor hver rad representerer en reaksjon. $g_r(\underline{\mathbf{c}})$ er en funksjon av reaktantene til reaksjonen. For å forklare Likning (25) så diskuter vi opp med et eksempel.

Eksempel: Tre reaksjoner

Si at vi har tre reaksjoner med tre reaksjonsrater:



Dette gir oss en $\tilde{\xi}$ vektor med tre elementer og en 3x3 matrise for $\underline{\underline{\mathbf{K}}}$. Fra reaksjonslikningen antar vi at reaksjonen bare går én vei, så reaksjonsraten er kun avhengig av konsentrasjonen til reaktanten. Vi setter opp de tre reaksjonlikningene for A, B og C:

¹I RekTek brukes notasjonen r_i for reaksjonsrater.

$$\tilde{\xi}_1 = k_1^r c_A \quad (26)$$

$$\tilde{\xi}_2 = k_1^r c_B \quad (27)$$

$$\tilde{\xi}_3 = k_1^r c_C \quad (28)$$

$$(29)$$

Med tre uttrykk for reaksjonratene setter vi dem sammen til et system av likninger:

$$\underline{\tilde{\xi}}(\underline{\mathbf{c}}) = \underline{\mathbf{K}} \underline{\mathbf{S}} \underline{\mathbf{c}} \quad (30)$$

$$\begin{bmatrix} \tilde{\xi}_1 \\ \tilde{\xi}_2 \\ \tilde{\xi}_3 \end{bmatrix} = \begin{bmatrix} k_1^r & 0 & 0 \\ 0 & k_2^r & 0 \\ 0 & 0 & k_3^r \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} c_A \\ c_B \\ c_C \\ c_D \end{bmatrix} \quad (31)$$

Likning (30) er det samme som Likning (25) hvorav vi har satt inn $\underline{\mathbf{S}} \underline{\mathbf{c}}$ for $g_r(\underline{\mathbf{c}})$.

Fra Likning (31) kan vi se at vi har et uttrykk for hver reaksjonrate.

7.4 Reaksjonsratekonstanten

Reaksjonsratekonstanten er ikke konstant, ironisk nok, men sterkt avhengig av temperatur. Ofte bruker man Arrhenius likning for å beregne denne “konstanten”. Arrhenius likning kjenner du sikkert igjen fra tidligere fag.

$$k_r^r(T) := k_r^0 e^{\frac{-E_A}{RT}} \quad (32)$$

k_r^0 = Pre-exponential factor

E_A = Aktiviseringsenergi

R = Universell gasskonstant

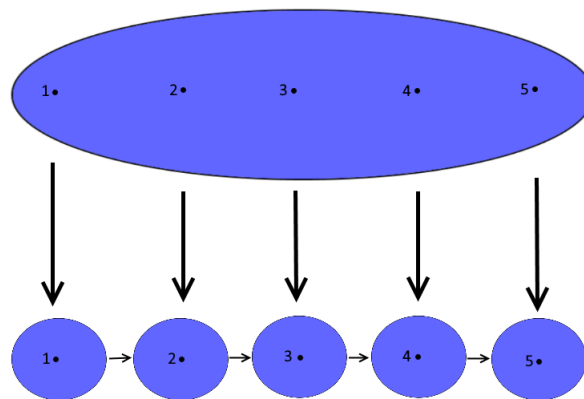
T = Temperatur

8 Approksimering av distributed systems

Et distributed system er et anisotropisk system med hensyn på intensive variabler. Det vil si at i et volum er, for eksempel, temperaturen avhengig av posisjonen i volumet. Som prosessingeniører må vi ofte lage en modell for denne anisotropiske oppførselen. Mange av disse modellene krever mye prosessorkraft og noen er rett og slett umulig å løse analytisk. I slike tilfeller må vi sette opp en approksimasjon for modellen. Her har vi flere metoder. Vi starter med den enkleste som har fått kallenavnet "Sliced Salami".

8.1 Sliced Salami method

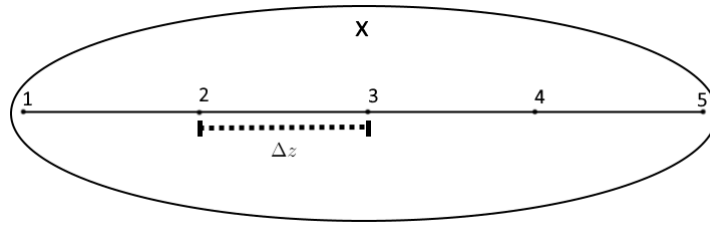
Sliced salami metoden er en måte å dele et distributed system inn i flere lumped systems. På den måten trenger vi ikke en modell for transporten gjennom det distributed systemet, men trenger istedet transporten mellom de mindre kontrollvolumene. Som du ser i Figur 10 er analogien at man tar salamien (Distributed system) og kutter det opp i flere mindre biter (lumped systems). Hver av disse systemene har egenskaper som tilsvarer posisjoner i det distributede systemet.



Figur 10: Visualisering av Sliced Salami Method. Et Distributed system deles opp i lumped systems.

8.2 Numerisk approksimasjon

En annen metode trekker deg tilbake til vårsemesteret og faget Matte 4N. Vi starter med å dele systemet vårt inn i et grid med punkter som spenner over rommet i de retninger hvor de intensive variablene varierer.



Figur 11: Distributed system med intensiv variabel x , delt opp i et 1-dimensjonalt grid med 5 punkter og steglengde Δz .

I Figur 11 ser vi at variabelen x endres fra posisjon 1 til posisjon 5. Vi har mange måter å approksimere endringen av x langs z . De fleste bygger på en Taylor utvidelse. Uten å bruke tid på å argumentere for hvorfor dette fungerer setter vi opp en Taylor utvidelse som evaluerer $i + 1$ fra posisjon i med lengde Δz :

$$x_{i+1} = x_i + \frac{dx_i}{dz} \Delta z + \frac{1}{2!} \frac{d^2 x_i}{dz^2} (\Delta z)^2 + \frac{1}{3!} \frac{d^3 x_i}{dz^3} (\Delta z)^3 + \dots \quad (33)$$

Litt skummel formel, men likevel nyttig å kunne (pst, dette er fra Matte 1 om du ikke husker). Som oftest i numeriske approksimasjoner benytter vi oss kun av de to første leddene:

$$x_{i+1} \approx x_i + \frac{dx_i}{dz} \Delta z \quad (34)$$

$$\downarrow \quad (35)$$

$$\frac{dx_i}{dz} \approx \frac{x_{i+1} - x_i}{\Delta z} \quad (36)$$

Nå som vi har en approksimering for $\frac{dx_i}{dz}$ kan vi sette denne inn for ukjente differensialer, f. eks. transportligninger. Approksimeringen vi har i ligningen over er navngitt *Forward differences* og er forholdsvis enkel å bruke. Under har vi lagt fram de vi mener er de viktigste numeriske metodene for approksimering av differensialer.

$$\begin{aligned} \text{Forward method: } \left. \frac{dx}{dz} \right|_i &\approx \frac{x_{i+1} - x_i}{\Delta z} \\ \text{Central differences 1st order: } \left. \frac{dx}{dz} \right|_i &\approx \frac{x_{i+1} - x_{i-1}}{2\Delta z} \\ \text{Central differences 2nd order: } \left. \frac{d^2 x}{dz^2} \right|_i &\approx \frac{x_{i-1} - 2x_i + x_{i+1}}{(\Delta z)^2} \end{aligned} \quad (37)$$

8.3 Numeriske skjemaer

Med den numeriske approksimasjonen kan vi substituere likningene fra (37) inn i en differensial likning slik at vi kan benytte oss av programmering til å løse problemet. Uten å bruke for mye tid og krefter på å forklare dette trekker vi fram et enkelt eksempel.

8.3.1 Eksempel: Euler's method

$$\frac{dx}{dz} = e^x + \ln(z) \quad (38)$$

Vi setter inn Forward differences metoden for differensialet så likningen kan løses eksplisitt for x_{i+1} .

$$\frac{x_{i+1} - x_i}{\Delta z} = e^{x_i} + \ln(z_i) \quad (39)$$

$$\downarrow \quad (40)$$

$$x_{i+1} = x_i + \Delta z(e^{x_i} + \ln(z_i)), \quad z_{i+1} = z_i + \Delta z \quad (41)$$

Dette kan vi implementere i python og løse alle x_i ved bruk av forrige x og z .

Istedet for å gjøre hele steget med substituering av Taylor ekspansjon, som vi gjorde over, kan du istedet formulere differensialet på formen:

$$\frac{dx}{dz} = f(x, z) \quad (42)$$

På denne formen kan vi benytte oss av forskjellige numeriske sjemaer som vist under. Siden numerikk ikke er en stor del av pensum i prossmod trekker vi bare fram to metoder for å løse første ordens differensialligninger.

$$\text{Eulers method: } x_{i+1} = x_i + \Delta z f(x_i, z_i) \quad (43)$$

$$\text{Heun's method: } x_{i+1} = x_i + \frac{1}{2} \Delta z [f(x_i, z_i) + f(x_{i+1}^p, z_{i+1}^p)] \quad (44)$$

$$x_{i+1}^p = x_i + \Delta z f(x_i, z_i) \quad (45)$$

Ekstra:

Hvorfor skal man bruke det ene skjemaet over det andre? Det har å gjøre med konvergeringen og nøyaktigheten til det numeriske skjemaet. Siden dette er approksimeringer vil den numeriske løsningen avvike fra den ekte løsningen. Avhengig av

problemet vi står overfor vil en metode være bedre enn en annen. I forskjellige tilfeller vil et skjema gi en rask løsning mens et annet skjema ikke vil gi en løsning i det hele tatt. Hvis du synes dette er nyttig og vil lære mer, anbefaler vi å ta faget TKT4140 Numeriske beregninger, senere i studiet.

9 Shell balance

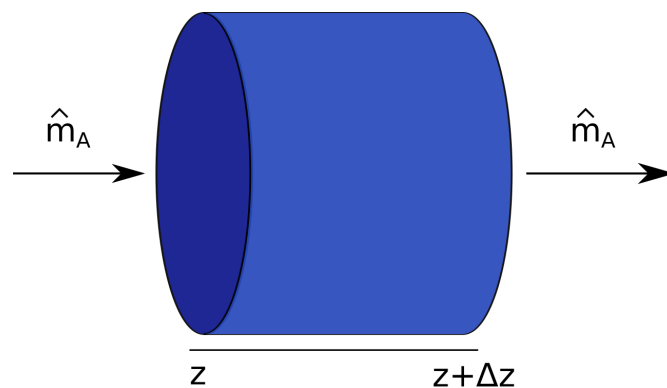
En skallbalanse eller “shell balance” er en måte å sette opp differensiallikninger som kan brukes til å beskrive hva som skjer i en prosess. Dette gjøres ved å utføre en balanse over en liten bit av system (en “unit cell”), typisk et lite volum. Herifra antar vi at balansen som gjelder over den lille biten av system også gjelder for hele systemet vi ser på. En slik balanse er basert på balansen vi introduserte i Avsnitt 5:

$$\text{Akkumulert} = \text{Strøm inn} - \text{Strøm ut} + \text{Generert} \quad (46)$$

Når vi setter opp likninger som beskriver det som skjer i en prosess, bruker vi ofte generelle likninger som vi forenkler (eg. Governing equation og Navier Stokes). Dette kan noen ganger være en ganske tungvint måte å gjøre det på, og en shell balance kan da være et godt alternativ for å finne fram til de nødvendige likningene.

9.1 Eksempel: Massestrøm gjennom en reaktor med reaksjon

La oss tenke oss at vi ser på en massestrøm gjennom en reaktor. Vi er da interessert i å se på hvordan systemet endrer seg langs reaktoren. Det første vi kan gjøre er å bestemme oss for en unit cell vi ønsker å gjøre balansen over. I dette tilfellet vil et naturlig valg av en slik unit cell være volumet mellom to tverrsnittareal i røret.



Figur 12: Shell balance over et lite volum i en reaktor. A kommer inn i posisjon z , reagerer gjennom reaktoren over lengden Δz og går ut i posisjon $z + \Delta z$.

La oss anta at følgende reaksjon skjer gjennom røret: $A \xrightarrow{\xi_A} B$

Videre, la oss sette opp balansen basert på komponent A (merk at vi også kunne satt opp balansen basert på komponent B). Ved å ta i bruk den generelle balansen

i Likning (46) kan en shell balance settes opp på følgende måte:

$$\overbrace{\frac{\partial m_A}{\partial t}}^{\text{Akkumulert}} = \overbrace{A \cdot \hat{m}_A|_z}^{\text{Inn}} - \overbrace{A \cdot \hat{m}_A|_{z+\Delta z}}^{\text{Ut}} - \overbrace{A \cdot \Delta z \cdot \xi_A}_{\Delta V}^{\text{Generert}} \quad (47)$$

Det kan hjelpe å se på benevningene til uttrykkene når du jobber med shell balance for å forstå hvordan du setter opp leddene. Merk at \hat{m}_A er fluksen av masse gjennom tverrsnittsarealet i z eller $z + \Delta z$. For at du ikke skal forvirres av likningen går vi gjennom hvert ledd:

$\frac{\partial m_A}{\partial t}:$	Endring av m_A inne i kontrollvolumet
$A \cdot \hat{m}_A _z:$	Transport av A inn til kontrollvolumet ved posisjon z ganget med tverrsnitt arealet (A)
$A \cdot \hat{m}_A _{z+\Delta z}:$	Transport av A ut av kontrollvolumet ved posisjon $z + \Delta z$ ganget med tverrsnitt arealet (A)
$A \cdot \Delta z \cdot \xi_A:$	Tap eller generering av A inne i kontrollvolumet ganget med volumet av unit cellen

Likningen vi har nå satt har et variabler som er definert i posisjon z og $z + \Delta z$. Som oftest har vi ikke kjennskap til posisjonen $z + \Delta z$ og vi må finne en måte å eliminere $\hat{m}_A|_{z+\Delta z}$ fra likningen vår. Vi har i hovedsak to metoder vi kan benytte oss for å få en løsbar likning:

- Definisjon av den deriverte
- Taylorrekke

Definisjonen av den deriverte går ut på å hente ut den matematiske definisjonen av derivasjon fra balansen så vi kan substituere differensiallikninger. I dette kompendiet har vi valgt å gå fram med Taylor utvidelse siden vi har alt brukt den i Avsnitt 8. Vi bruker en Taylorrekke på Likning (47) til å approksimere $\hat{m}_A|_{z+\Delta z}$. Dette kan gjøres ved å approksimere posisjon $z + \Delta z$ ved bruk av den deriverte i posisjon z . Når vi bruker en Taylorrekke til dette, benytter vi oss av samme metode som i Avsnitt 8.2 og bruker kun de to første leddene i rekken, til og med den første deriverte:

$$\hat{m}_A|_{z+\Delta z} \approx \hat{m}_A|_z + \left. \frac{\partial \hat{m}_A}{\partial z} \right|_z \cdot \Delta z \quad (48)$$

Ved å kombinere Likning (47) og Likning (48) blir balansen enklere å jobbe med siden alle variablene er i posisjon z . Notasjonen $|_z$ sløyfes herfra for enkelhetens

skyld.

$$\frac{\partial m_A}{\partial t} = A \cdot \left(\hat{m}_A - \hat{m}_A - \frac{\partial \hat{m}_A}{\partial z} \cdot \Delta z \right) - \Delta V \cdot r_A \quad (49)$$

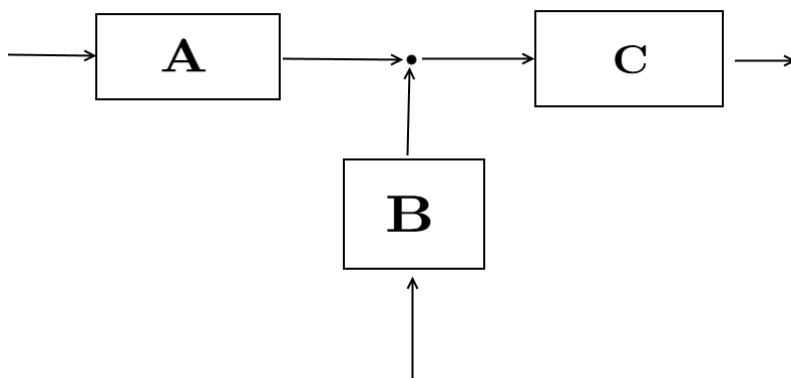
Her bør det merkes at $\Delta V = A \cdot \Delta z$. Herifra kan vi forenkle likningen, for så og dele på ΔV . Ved å la $\rho_A = m_A/V$ får vi en massebalanse gjennom røret:

$$\frac{\partial \rho_A}{\partial t} = \frac{\partial \hat{m}_A}{\partial z} - r_A \quad (50)$$

I dette eksemplet burde det merkes at $r_A [=] \frac{kg}{s \cdot m^3}$. For en (litt) mer detaljert forklaring på hvordan vi går fra likning Likning (49) til Likning (50) anbefales det å se ABC-heftet Avsnitt 4.

10 Blokkdiagrammer og The Grand Scheme

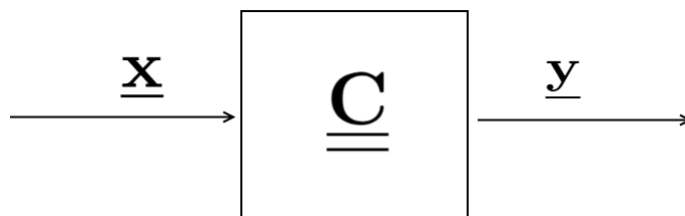
I alle fag frem til nå har du sett på et flytdiagram med piler og bokser og tenkt at dette er masse som beveger seg fra en tank til en annen. Ta eksempelet gitt i Figur 13. Hadde vi spurt en kybernetiker hadde han beskrevet figuren som en oversikt over informasjonsflyten til systemet. Som prosessingeniør er det viktig å vite forskjellenene mellom blokkdiagram og flytskjema og deres respektive bruksområder.



Figur 13: Et blokkdiagram.

10.1 Blokkdiagram

Et blokkdiagram er en visualisering av signaler som sendes fra et system til andre systemer. Til forskjell fra flytskjema er pilene i et blokkdiagram signaler som bærer informasjon istedet for masse. Ved å bruke et blokkdiagram kan vi visualisere hvordan systemer kommuniserer med hverandre i form av lineære sammenhenger.

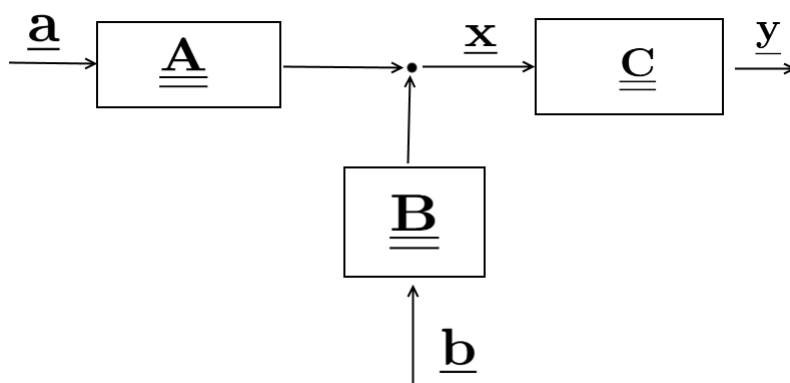


Figur 14: Et veldig enkelt blokkdiagram

Blokkdiagrammet i Figur 14 viser den lineære sammenhengen mellom \underline{x} og \underline{y} , som kan beskrives som ligningen:

$$\underline{y} = \underline{C} \underline{x} \quad (51)$$

Trikset med blokkdiagrammer er at alle variablene/vektorene/matrisene ganges med hverandre via en pil og plusses sammen hvis pilen møter en annen pil. Det er mange måter å angripe et blokkdiagram, men en enkel metode er å starte bakerst og bevege deg mot pilens retning. Da er du sikker på at du får riktig plassering på vektorer og matriser.



Figur 15: Blokk diagram med 3 systemer hvor signalet \underline{y} er en lineær kombinasjon av signalene \underline{a} og \underline{b} .

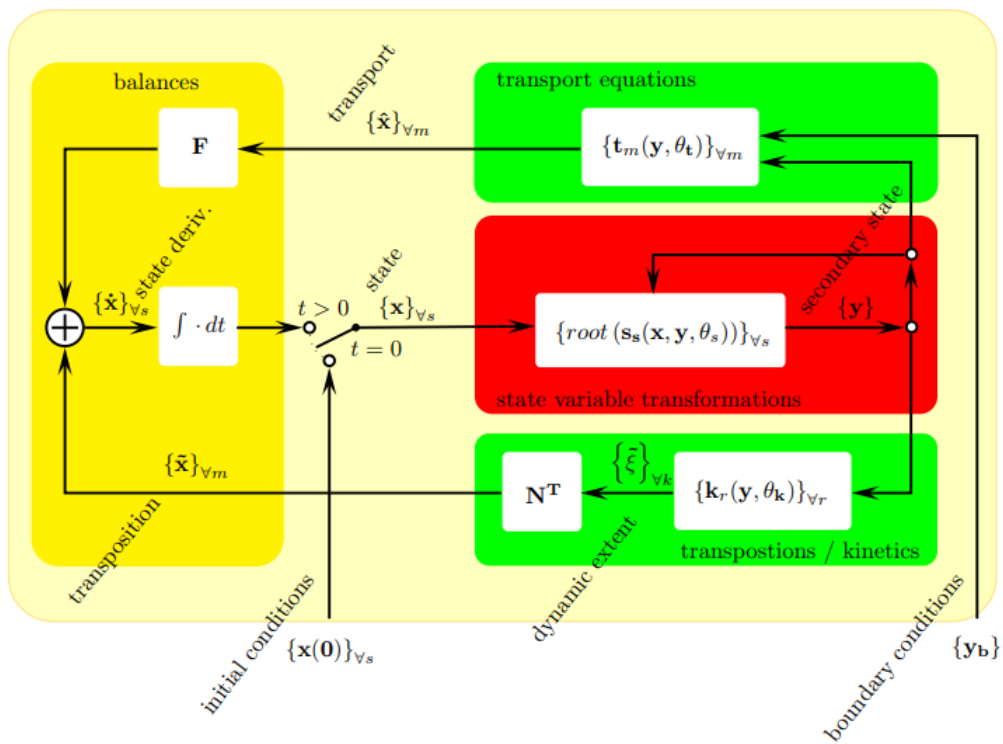
Fra Figur 15 er det to likninger vi kan hente ut. Den første er den samme som i Likning (51) og den andre er den lineær kombinasjonen mellom $\underline{A}\underline{a}$ og $\underline{B}\underline{b}$:

$$\underline{y} = \underline{C}\underline{x} \quad (52)$$

$$\underline{x} = \underline{A}\underline{a} + \underline{B}\underline{b} \quad (53)$$

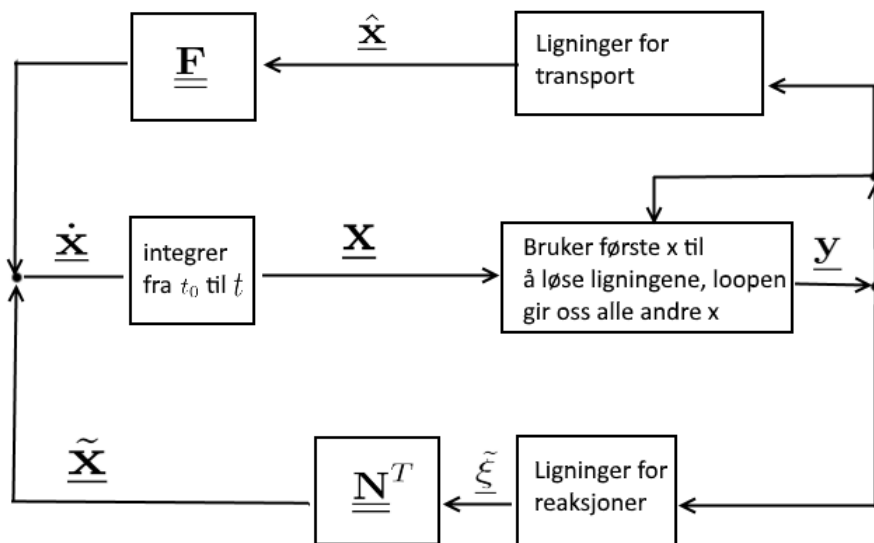
10.2 The Grand Scheme

The Grand Scheme er hele pensum summert opp. Tanken bak The Grand Scheme er at man kan koble sammen alle delene i modellering til et blokkdiagram. I ABC-heftet blir The Grand Scheme forklart ved et blokkdiagrammet gitt i Figur 16:



Figur 16: The Grand Scheme, hentet fra ABC-heftet.

Siden dette blokkdiagrammet er litt vanskelig å skjønne har vi laget en versjon som er litt mindre korrekt, men litt mer lesbar når man først introduseres til The Grand Scheme.



Figur 17: The Grand Scheme, forenklet.

Bruker vi det vi har lært om blokkdiagrammer og tyder Figur 16 i form av linære kombinasjoner får vi et sett med ligninger som beskriver det essensielle i prossmod. Under er f, g, h funksjoner for henholdsvis state, reaksjoner og transport.

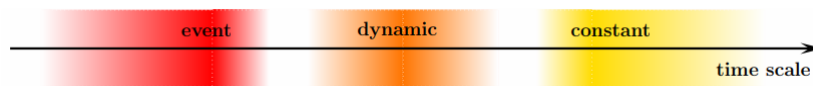
$$\begin{aligned}
 \dot{\underline{x}} &= \underline{\underline{F}} \hat{\underline{x}} + \tilde{\underline{x}} \\
 \tilde{\underline{x}} &= \underline{\underline{N}}^T \tilde{\underline{\xi}} \\
 \tilde{\underline{\xi}} &= g(\underline{y}) \\
 \hat{\underline{x}} &= h(\underline{y}) \\
 \underline{x}_{\text{next state}} &= f(\underline{y})
 \end{aligned} \tag{54}$$

Likningene over kjenner du forhåpentligvis igjen fra tidligere kapitler. Det er ikke så mye mer å si om The Grand Scheme enn at mesteparten av pensum kan summeres opp i det.

11 Reduksjon av modeller med time scale og kapasitet

En modell er alltid bygget på et sett med antagelser som divergerer modellen fra det reelle. Frem til nå har vi gjort antagelser på kapasitet og time scale uten å være fullt klar over argumentene for antagelsene. Time scale assumptions er kort fortalt neglisjering av modeller pga dynamiske forskjeller. Vi kan dele en time scale inn i tre deler:

- Event dynamic: Skjer umiddelbart
- Dynamisk: Forandring over tid som er merkbart
- Konstant: Ingen forandring i system



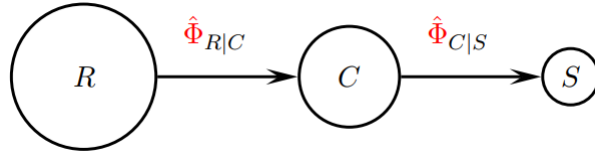
Figur 18: Time scale som visualiserer hvordan event, dynamic og constant henger sammen. Merk at ingen av domenene overlapper.

Vi har mange måter å gå fram på for å redusere modeller. Vi bruker i hovedsak 4 angrepsvinkler. De fire forskjellige metodene blir nærmere forklart i det kommende eksempelet i Avsnitt **11.1**.

1. Ekspandere det som er konstant
2. Introdusere event dynamic
3. Analysere den raske dynamikken
4. Analysere den trege dynamikken

11.1 Eksempel: Tre innsjøer

Vi ønsker å modellere vannet som renner fra en stor innsjø (R) til en mindre (C) og videre til en enda mindre innsjø (S). For å forklare reduksjonen best mulig trekker vi fram en enkel topologi:



Figur 19: Topologi for tre innsjøer hvor R transporterer til C som transporterer til S.

Den fulle modellen for topologien for Figur 19 gir likningene:

$$\dot{\Phi}_R = -\hat{\Phi}_{R|C} \quad (55)$$

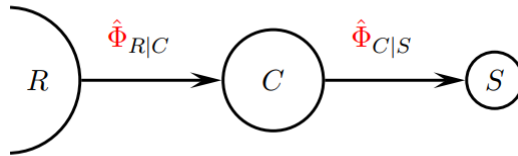
$$\dot{\Phi}_C = \hat{\Phi}_{R|C} - \hat{\Phi}_{C|S} \quad (56)$$

$$\dot{\Phi}_S = \hat{\Phi}_{C|S} \quad (57)$$

Vi gjør reduksjon ved bruk av de fire angrepsvinklene. I ABC-heftet kapittel 16 har de en matematisk forklaring på reduksjonen, den har vi valgt å ikke ta med til fordel for den intuitive forklaringen.

1: Ekspandere det som er konstant.

Ved å anta at R er mye større enn C og S så kan vi observere R som et reservoar som fører til at vi ikke trenger å modellere R ($\dot{\Phi}_R = 0$).



Figur 20: Topologi for tre innsjøer hvor R er betydelig mye større enn C og S.

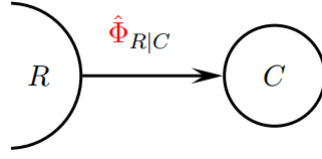
Dette gir oss en ny modell:

$$\dot{\Phi}_C = \hat{\Phi}_{R|C} - \hat{\Phi}_{C|S} \quad (58)$$

$$\dot{\Phi}_S = \hat{\Phi}_{C|S} \quad (59)$$

2: Introdusere event dynamic.

Hvis vi antar at volumet S er minimalt forhold til C, kan vi anta at det som går ut av C til S er neglisjerbart forhold til transporten fra R til C.



Figur 21: Topologi for tre innsjøer hvor transporten fra C til S er så minimal at det ikke har noen påvirkning på modellen.

3: Analysere den raske dynamikken (small time scale).

Vi tar fram topologien i Figur 20, etter ekspansjonen av det konstante domenet, og setter inn en lineær modell for transport som vi lærte i Avsnitt 6.4.

$$\dot{\Phi}_C = \hat{\Phi}_{R|C} - \hat{\Phi}_{C|S} \quad (60)$$

$$\dot{\Phi}_S = \hat{\Phi}_{C|S} \quad (61)$$

$$\dot{\Phi}_C = -k_{R|C}(\pi_C - \pi_R) + k_{C|S}(\pi_S - \pi_C) \quad (62)$$

$$\dot{\Phi}_S = -k_{C|S}(\pi_S - \pi_C) \quad (63)$$

Ved å anta at transporten $\hat{\Phi}_{R|C}$ er mye tregere enn $\hat{\Phi}_{C|S}$ kan vi anta at ved en liten time scale vil $\hat{\Phi}_{C|S}$ dominere og vi kan neglisjere transporten fra R til C i $\dot{\Phi}_C$:

$$\dot{\Phi}_C = +k_{C|S}(\pi_S - \pi_C) \quad (64)$$

$$\dot{\Phi}_S = -k_{C|S}(\pi_S - \pi_C) \quad (65)$$

4: Analysere den trege dynamikken (large time scale).

Å bruke en lang time scale er veldig vanlig i prosessmodellering. Etter en lang tid vil forskjellen i effort variabler, mellom de to systemene, forsvinne. Tenk deg at du har laget deg en kopp med varm kaffe. Venter du i 24 timer før du drikker kaffen kan du forvente at temperaturen til koppen er den samme som rommet.

$$0 = k(\pi_S - \pi_C) \quad (66)$$

$$\pi_C = \pi_S \quad (67)$$

12 Programmering

Programmering i prosessmodellering er utfordrende på flere plan. For det første er det forventet at du kan en del av det du har lært i ITGK, Prosessteknikk og tidligere mattefag. Hvis du ikke husker disse metodene anbefaler vi deg å lese deg opp på dette før du begynner på programmeringsdelen. I prossmod er det forventet at du skal kunne bruke numerikk til å løse sett med ODE (Ordinary Differential Equations). I tillegg er det forventet at diverse problemer av skal gjøres objektorientert. Dette kan være utfordrende og krever en del tid, men vi skal prøve å guide deg gjennom de viktigste punktene i objektorientert programmering. Hvis du greier å lære deg dette har du et stort fortrinn til vårfaget TDT4102 - Prosedyre- og objektorientert programmering (C++), som vi anbefaler enhver student på IKP å ta.

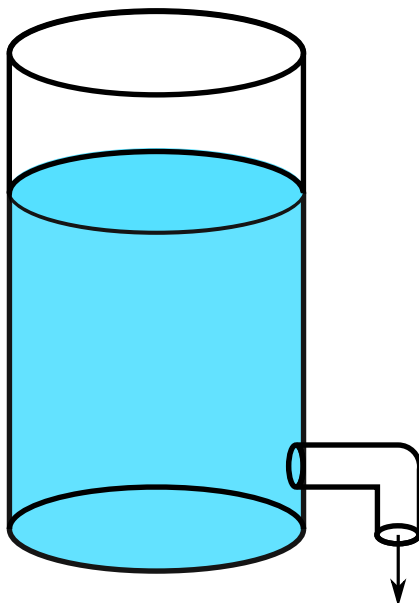
12.1 Løse ODE med programmering

Dere har alle lært dere og løse differensiallikninger analytisk i Matte 1. Da analytiske løsninger ikke var tilgjengelig lærte dere så å løse dem numerisk i Matte 4. Fra Avsnitt 8.2 lærte vi hvordan man approksimerte en differensiallikning med bruk av Taylor utvidelser. Tilstanden i punkt $i + 1$ kan ofte beregnes hvis vi vet tilstanden i punkt i . Det kan imidlertid bli fryktelig slitsomt å regne 20 iterasjoner med Heuns metode eller Runge Kutta for hånd. Enda verre blir det om dere har et system av differensiallikninger og må løse 20 iterasjoner for tre eller flere likninger. Det er da utrolig nyttig at dere kan gjøre dette på en PC, og det er nettopp dette vi skal ta for oss nå.

Eksempel: Tømme en tank

Vi har en tank med utløp som har parameterne gitt i tabell Tabell 2.

Tabell 2: Parameterverdier for tømmeprosessen.				
Parameter	$d_{\text{rør}}$ [m]	d_{tank} [m]	g [m s^{-2}]	$h(0)$ [m]
Verdi	0,1	1	9,81	2



Figur 22:

1. Sett opp en ligning som viser høyden i tanken som funksjon av tid
2. Bestem hastigheten for å sette ligningen på løsbar form
3. Skriv et program som løser differensialligningen og plotter høyden som funksjon av tid

Løsningsforslag til punkt 1:

Vi ønsker å finne ut hvor lang tid det tar å tømme den åpne tanken med vann. For å gjøre dette må vi først sette opp en enkel modell for systemet. Ved å bruke den generelle Likning (2) for massebevaring setter vi opp konserveringslikningene for masse som tidligere:

$$\dot{m} = m_{\text{inn}} - m_{\text{ut}}, \quad (68)$$

der \dot{m} er hvordan massen i tanken endrer seg, mens m_{inn} og m_{ut} er henholdsvis massestrømmer inn og ut av tanken. Siden vi har interesse av å finne høyden til

massen kan vi omformulere (68) i form av volum:

$$\frac{d(\rho V)}{dt} = (\rho q)_{inn} - (\rho q)_{ut}, \quad (69)$$

der ρ er massetettheten og q er volumstrømmen. Siden massetettheten til vann er konstant, kan denne trekkes ut av den deriverte og strykes på begge sider:

$$\frac{dV}{dt} = q_{inn} - q_{ut} \quad (70)$$

Siden tanken vår kan beskrives som en sylinder kan vi sette inn volumet for en sylinder $V = \pi r^2 h$. Når tanken tømmes vil væsken oppta et mindre volum av tanken, og det er høyden som endres etter hvert som tiden går. Det betyr at vi kan trekke πr^2 ut av den deriverte slik at

$$\frac{dh}{dt} = \frac{1}{\pi r^2} (q_{inn} - q_{ut}) \quad (71)$$

Til slutt vil vi fjerne q_{inn} siden det kun går masse ut av tanken, og det er da ingen volumstrøm inn. Merk at dette leddet kunne blitt fjernet tidligere i utledningen. Nå sitter vi igjen med en differensiallikning for endringen av høyden til tanken som funksjon av kun t og q_{ut} .

$$\frac{dh}{dt} = -\frac{1}{\pi r^2} q_{ut} \quad (72)$$

Løsningsforslag til punkt 2:

Vi vet fortsatt ingenting om q_{ut} , så vi kan enda ikke løse (72). For enkelhetens skyld benytter vi oss av hastighetsuttrykket for rørstrømning

$$q = vA, \quad (73)$$

der v er hastigheten og A er tverrsnittsarealet til røret. Deretter tar vi for oss Bernoulli som er kjent fra strømning fra våren i fjor

$$\frac{1}{2}v_1^2 + gz_1 + \frac{p_1}{\rho} = \frac{1}{2}v_2^2 + gz_2 + \frac{p_2}{\rho} \quad (74)$$

Vi velger å se på en strømlinje fra toppen av vannsøylen der hastigheten er 0 og trykker er $p_0 = 1 \text{ atm} \approx 1 \text{ bar}$. I tillegg er trykket ved utløpet også $p_0 \approx 1 \text{ bar}$, siden vannet som kommer ut av utløpet også er i kontakt med lufta rundt. Ved å sette nullnivå ved utløpet og $z_1 = h$ er utløpshastigheten dermed gitt av

$$v_2 = \sqrt{2gh} \quad (75)$$

Ved å sette (75) inn i (73), og deretter sette resultatet inn i (72) får vi følgende uttrykk

$$\frac{dh}{dt} = -\frac{1}{\pi r_{tank}^2} \sqrt{2gh} \frac{\pi}{4} d_{rør}^2 \quad (76)$$

Vi pynter litt på uttrykket og ender opp med den implementerbare likningen for tømmeprosessen

$$\frac{dh}{dt} = -\left(\frac{d_{rør}}{d_{tank}}\right)^2 \sqrt{2gh} \quad (77)$$

Løsningsforslag til punkt 3:

For å løse differensiallikningen med programmering bruker vi parameterene gitt i tabell 2. Likning (77) kan beskrives som en første ordens differensiallikning og som vist i Avsnitt 8.2 kan vi approksimere en differensiallikning ved bruk av Taylor rekker.

```

1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  # ===== IMPORTS ===== #
5  import numpy as np
6  import matplotlib.pyplot as plt
7
8
9  # ===== PARAMETERS ===== #
10 d_pipe = 0.1          # Diameter for utl p s r r
11 d_tank = 1            # Diameter for tanken
12 g = 9.81             # Gravitasjonskonstant
13 h0 = 2               # Starth yde
14 dt = 0.1             # Steglengde
15 t0 = 0               # Starttid
16 tN = 60              # Sluttid
17 N = int((tN-t0)/dt + 1) # Antall punkter
18
19
20 # ===== FUNCTION DEFINITIONS ===== #
21 def dhdt(h, t):
22     # H yresiden av likningen y' = f(h, t), med andre ord f(h, t)
23     return -(d_pipe/d_tank)**2 * np.sqrt(2*g*h)
24
25 def euler(dt, h, t):
26     # Numerisk finner neste y ved y_{n+1} = y_n + h * f(h, t)
27     return h + dt * dhdt(h, t)
28
29
30
31
32

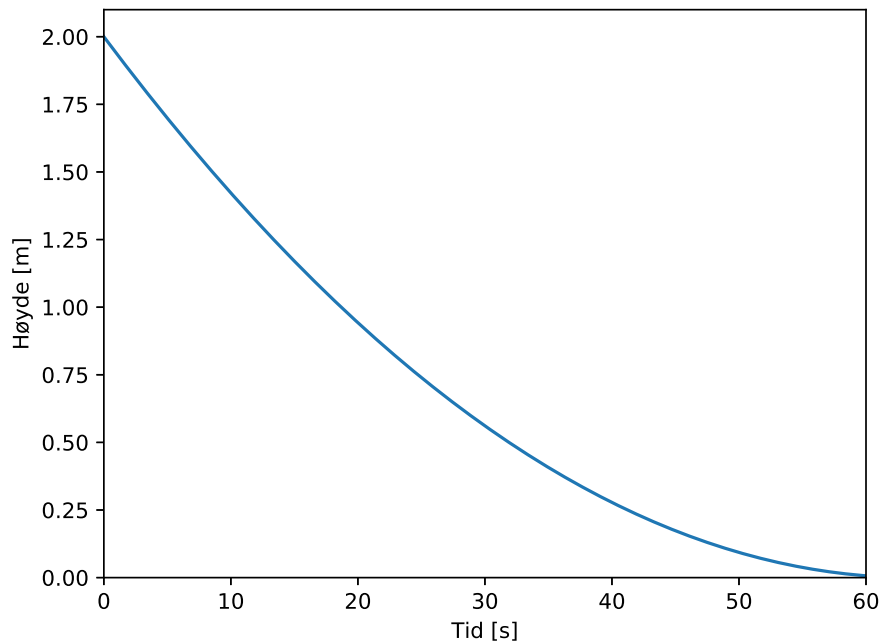
```

```

33
34 def solve_ode():
35     # Sett starttiden, det vil si initialtid
36     t_current = t0
37
38     # Lag vektorer som lagrer l sningen og tiden
39     h = [h0]
40     t = [t0]
41     for _ in range(N):
42         t_current += dt
43         h_current = h[-1] # Hent siste element
44         h_new = euler(dt, h_current, t) # Beregn neste h yde
45         # Legg til tid og h yde i vektorer
46         t.append(t_current)
47         h.append(h_new)
48     # Lag grafisk representasjon av h yden som funksjon av tid
49     plt.plot(t, h)
50     # Juster aksene
51     plt.xlim(t0, tN)
52     plt.ylim(0, 2.1)
53     # Lag aksetitler
54     plt.xlabel('Tid [s]')
55     plt.ylabel('H yde [m]')
56     # Lagre bildet i h y kvalitet
57     plt.savefig('tank.eps', format='eps', dpi=400)
58     # Vis plottet :)
59     plt.show()
60     plt.close()
61
62
63 # ===== PROGRAM EXECUTION ===== #
64 if __name__ == '__main__':
65     solve_ode()

```

Når vi kjører koden så får vi ut plottet i Figur 23.



Figur 23: Plott av tankens høyde som funksjon av tiden.

12.2 Klasser og objekter

Til nå har dere ofte programmert i en fil med definering av variabler øverst i filen og så skrevet viktige funksjoner under. I IT-verdenen er dette kronglete siden et program kan trenge flere titusen linjer med koder så vi må ha et system for å sortere koden. Samtidig vil operasjoner i programmet gjenta seg og vi ønsker ikke å skrive en kode flere ganger. Av den grunn er objektorientert programmering fantastisk siden vi kan generalisere hvordan en del av koden skal oppføre seg i forhold til en annen del. Formelt sett er klasser et overordnet rammesystem for hvordan objekter kommuniserer sammen. Det kan være alt fra hvordan en bil ser ut til hvordan et matematisk system fungerer. Før dere blir mer forvirret trekker vi fram et eksempel på klasser og objekter.

12.2.1 Eksempel: Katter og hunder

Tenk deg at du har lager et dataspill med en åpen verden og sjefen din ber deg om å fylle verdenen med katter. Du tenker deg om og kommer fram til følgende konklusjon: Alle kattene i denne verdenen har et navn og har fire bein. Istedet for å programmere dette for hver eneste katt lager vi et generelt rammeverk som gjelder

alle katter.



Figur 24: En visualisering av klassen "Katt".

Når vi lager et rammeverk for alle kattene kan vi fylle verdenen vår med mange katteobjekter. Det vil si at i verdenen vår har vi mange forskjellige katte-objekter, men alle katte-objektene tilhører klassen "Katt" og hvert katte-objekt vil ha et navn og fire bein. Ved bruk av python som programmeringspråk skriver vi klassen for katt.

Del 1 (Kode):

```
1 class Katt: #Klassen katt
2     #Medlemsvariabler
3     antall_ben = 4
4     navn = "" #Default navn
5
6     #Constructor
7     def __init__(self, navn): #Lager Katte-objektet med spesifisert navn
8         self.navn = navn #Endrer navnet til katten fra "" til det vi spesifiserte
```

Del 2 (Konsoll):

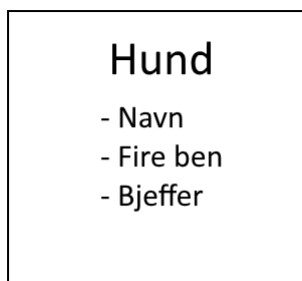
```
1 >>> katten_butte = Katt("Butte") #Lager et katteobjekt med variabelnavn "
   katten_butte"
2 >>> print(katten_butte.navn)
3 Butte
4 >>> print(katten_butte.antall_ben)
5 4
```

En liten forklaring på hva som skjer.

"__init__(self,navn)" er en konstruktør for et katte-objekt. init kommer fra det engelske ordet "initial". Når vi skal lage et et katteobjekt må vi kalle på konstruktøren vår. Ved å kalle på konstruktøren lager python et tomt objekt og sender det til oss i form av "self". Self er på mange måter "Seg selv" så når vi bruker self snakker

vi om det spesifikke katte-objektet. Konstruktøren tar også inn et navn som vi må spesifisere. Vi kunne ha programmert konstruktøren til å ta inn antall bein også, men siden en katt alltid har 4 bein så gir det ingen mening å la brukeres bestemme hvor mange bein et katte-objekt skal ha. På linje 1 i Del 2 lager vi et objekt som vi kaller "katten_butte" som vi spesifiserer skal ha navnet "Butte". Navn og antall bein til katten er en instans av klassen vår. Det vil si "Egenskaper" til katte-objektet. Her er det viktig å skille mellom variabelnavnet på objektet og instansen til objektet som er vi har valgt å kalle "navn".

Sjefen vår er ikke helt fornøyd med at spillverdenen bare består av katter. Han ber oss om å legge til hunder i tillegg til kattene og hundene skal kunne bjeffe.



Figur 25: En visualisering av klassen "Hund".

Hunden har de samme medlemsvariablene som en katt, men skal i tillegg ha medlemsfunksjonen å bjeffe. Dette gjør vi ved å lage det som kalles en medlemsfunksjon til klassen. En funksjon som bare klassen "Hund" kan benytte seg av.

Del 1 (Kode):

```
1 class Hund: #Klassen Hund
2     #Medlemsvariabler
3     antall_ben = 4
4     navn = "" #Default navn
5
6     #Constructor
7     def __init__(self, navn): #Lager hunde-objektet med spesifisert navn
8         self.navn = navn #Endrer navnet til hunden fra "" til det vi spesifiserte
9     #Medlemsfunksjon
10    def bjeffe(self):
11        print("BARK BARK")
```

Del 2 (Konsoll):

```
1 >>> Hunden_tara = Hund("Tara") #Lager et Hundeobjekt med variabelnavn "Hunden_tara"
```

```
2 >>> print(Hunden_tara.navn)
3 Tara
4 >>> Hunden_tara.bjeffe() #Kaller bjeffefunksjonen
5 BARK BARK
```

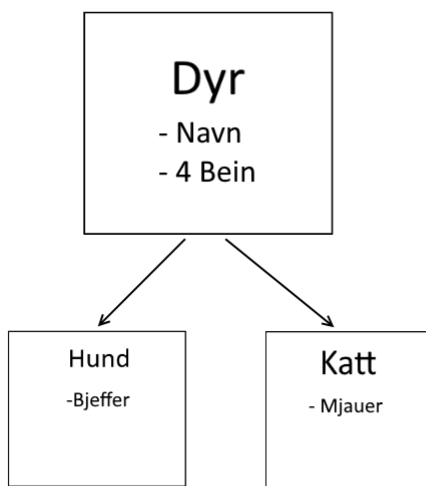
Nå har vi en klasse for Hund og for Katt som alle objekter i spillverdenen vår må følge. En klasse er et strengt rammeverk i form av at et objekt ikke kan gjøre annet enn det klassen tillater. Det vil si at hvis vi prøver å skrive `katten_butte.bjeffe()` så får vi feilmelding siden klassen Katt, som `katten_butte` er et objekt av, ikke har en funksjon som heter `bjeffe()`.

12.3 Superklasser, subklasser og arv

I mange tilfeller vil man ha klasser som har de samme medlemsvariablene og samme medlemsfunksjonene. Istedet for å definere de samme funksjonene til hver klasse er det smartere å la en klasse arve medlemsfunksjoner og medlemsvariabler fra det vi kaller en superklasse.

12.3.1 Eksempel: Mange forskjellige dyr

Sjefen vår kommer inn og sier at vi også skal ha kuer, griser, hester og mange andre forskjellige dyr i spillverdenen vår. Vi oppdager at dette blir mye repetisjon av kode som vi må spesifisere for hver klasse. Hund og katt hadde begge 4 ben og et navn. Her kan vi koble disse to klassene til en superklasse som vi kaller Dyr. Hund og Katt blir da subklasser av superklassen Dyr.



Figur 26: En visualisering av superklassen Dyr med Katt og Hund som subklasser.

```

1 class Dyr(): # Klassen Dyr
2     #Medlemsvariabler for alle dyre-objekter
3     antall_ben = 4
4     navn = ""
5     #Constructor for dyre-objekter
6     def __init__(self, navn):
7         self.navn = navn
8
9 class Katt(Dyr): # Klassen Katt, arver fra klassen Dyr
10     #Medlemsfunksjoner
11     def mjaue(self):
12         print("MJAAAU")
13
14     def __init__(self, navn): #Spesifiserer konstruktør for Katt-klassen
15         super().__init__(self, navn) #Kaller superklassen sin constructor
16
17
18 class Hund(Dyr): # Klassen Hund, arver fra klassen Dyr
19     #Medlemsfunksjon
20     def bjeffe(self):
21         print("Walla bruh, bjeff litt for meg da")

```

Som du ser i koden over så har klassen Katt en spesifisert konstruktør mens Hund har ingen. I realiteten er det ingen forskjell mellom klassene siden linje 15 kaller på konstruktøren til Dyr hvorav Katt vil automatisk kalle på konstruktøren. Denne kalles automatisk siden Katt arver konstruktøren til klassen Dyr. Vi valgte å ta med linje 14 og 15 for å vise deg hvordan du kan definere din egen konstruktører for subclasser. Dette er nyttig å kunne når subclasser skiller seg fra superklasser.

Ekstra:

Under er koden for filen "Min_python_fil.py"

```

1 if __name__ == "__main__":
2     print("Dette er hovedskriptet")

```

Linje 1 i koden over er litt merkelig å forstå, men tanken er at hvis du kaller på den spesifikke filen "Min_python_fil.py" så vil betingelsen til if-setningen gi ut True som gjør at linje 2 vil kjøre.

```

1 >>> python Min_python_fil.py
2 Dette er hovedskriptet

```

Vi ønsker å gjøre dette er fordi filen kan importeres fra andre filer. Hvis vi kjører en annen fil som importerer "Min_python_fil.py", så vil ikke if-betingelsen godkjennes.

13 Tabell med de viktigste ligninger

Under har vi ramset opp de likningene vi mener du bør huske til eksamen. Dette gjelder ikke nødvendigvis likninger som du har blitt introdusert for i tidligere fag. I tillegg er det viktig å påpeke at noen likninger, som shell balance, må du lære deg å utlede siden å pugge dem vil sjelden treffe riktig.

Totalt akkumulering	$\Phi = \int_{t_0}^t \dot{\Phi} dt$
Balanse	$\dot{\Phi} = \hat{\Phi}_{inn} - \hat{\Phi}_{ut} + \tilde{\Phi}$
Konservering	$\dot{\Phi} = \hat{\Phi}_{inn} - \hat{\Phi}_{ut}$
System av transportligninger	$\underline{\dot{\Phi}} = \underline{\mathbf{F}} \underline{\hat{\Phi}}$
Transport av ekstensiv variabel	$\hat{\varphi} = -c \frac{\partial \pi}{\partial \mathbf{r}}$
Lineær transportligning	$\hat{\Phi}_{a b} = -k_{a b}(\pi_a - \pi_b)$
Generering av stoffmengde	$\underline{\tilde{n}} = V \underline{\mathbf{N}}^T \underline{\tilde{c}}(\underline{\mathbf{c}})$
Kjemisk potensial	$\mu_i = \mu_i^0 + RT \ln(x_i)$
Reaksjonsrate	$\tilde{\xi} = k_r^r g(c)$
Volum til masse	$m = \frac{m}{\rho}$
Bernoulli uten tap	$\frac{1}{2}v_1^2 + gz_1 + \frac{p_1}{\rho} = \frac{1}{2}v_2^2 + gz_2 + \frac{p_2}{\rho}$
Arrhenius ligning	$k_r^r(T) = k_r^0 e^{\frac{-E_A}{RT}}$
Taylor utvidelse av første orden	$x_{i+1} \approx x_i + \frac{dx_i}{dz} \Delta z$
Volum av en sylinder	$V = \pi r^2 h$
Volum av en kule	$V = \frac{4}{3}\pi r^3$
Hastighet i rørstrømning	$q = vA$

A Referanse

Heinz A. Preisig, *ABC of modelling*, hentet fra TKP4106 <http://folk.ntnu.no/preisig/>