Reaktiv beregningsmekanikk – en løsningsstrategi

Teodor Heggelund

2017-05-13

Innhold

1 Symptomer

Tradisjonell ingeniørvitenskap er basert på matematikk, mekanikk og forankring i standarder. Årsakssammenheng er stort sett godt forklart. I verste fall ser vi på en litt rar faktor standarden som har blitt kalibrert etter numeriske forsøk. Det er uansett mulig å framstille funksjonen mellom inputparameterene og kapasiteten grafisk og få en mageføelse.

Problemene vi møter har blitt mer kompliserte og mer komplekse i senere tid. Lag på lag er bygget over problemet så det skal være mulig å få *resultatet* av en beregning. FEM kan løse statikken for oss. Forutsatt er at hva som skjer i alle lagene mellom ikke er så farlig, så lenge resultatet er en utnyttelse mindre enn 1. Forutsatt er at inputparameterne er perfekt forstått. Forutsatt er at det er null usikkerhet i inputparametre. Å, du sier det er usikkerhet i en inputparameter? Da trenger vi faktorer. Så det er vanskelig å si om faktoren skal øke eller senke en verdi for å oppnå et konservativt resultat? Kjør dobbelt opp med lastkombinasjoner. Du snakker om *stivhet*, som ikke kommer med i en lastkombinasjon? Nei, det sier vi ikke noe om. Anta noe, du, det går sikkert greit. Ikke komfortabel med å anta? Nei, da får du heller

kjøre to separate analyser. Antallet kombinasjoner eksploderer eksponentielt. Verktøy som omhylningsverdier blir introdusert, og vi ser ikke lenger på resultatet av én analyse, men en myriade av forskjellige forutsetninger som kun skal brukes til å spesifisere armeringsbehovet i alle dekker og skiver. Hva er det som forårsaker dette da? Ikke programvarens ansvar.

Forutsetningen om at input og problemet faktisk kan forstås knuses. Direkte rasjonalitet erstattes av mageølelse. Mange sitter med erfaring med så mange gjennomførte prosjekter at denne mageølelsen er *meget* god. Andre har knapt med erfaring og et mystisk forhold til FEM.

Hva skjer så i prosjekter der årsakssammenhengen er avgjørende for forståelsen av konstruksjonens oppførsel? Hvordan bygger vi konstruksjoner som er utenfor alt vi tidligere har gjort, når vi ikke lenger kan stole på de eldres mageølelse? Vi må lage våre egne verktøy. Helt ny teori utvikles parallellt i alle store, kompliserte prosjekter. Individer blir uvurderlige for deres hardt tilegnede systemkunnskap, og **må** være med på prosjektene. Tidspress begrenser mulighet til å ta et steg tilbake og generalisere, og det som kanskje kunne blitt gull må gjenoppdages i neste prosjekt.

Vi sitter i dag på verktøy som lar oss *ignorere* detaljene, under forutsetning om at de er uvesentlige. Detaljene er grunnen til at analyse fungerer i vitenskap. Detaljene er vesentlige.

2 Underliggene problem

Hvorfor har det blitt sånn? Brukergrensesnitt krever et minste felles multiplum. Detaljer er vanskelige å forklare. Gode forsøk blir gjort isolert, på vidt forskjellig måte, og må gjenoppdages fra program til program.

Vi mangler språk til å snakke om detaljene.

Språket må ligge nær detaljene og være lærbart. Når vi ønsker å løse et spesialtilfelle bør språket oppmuntre oss til å se den generelle løsningen. Mange gode programmer for elementanalyse er lærbare, men kan da ikke la oss se detaljene i løsningen. Programmering er helt generelt, men langt ifra lærbart og reflekterer en helt annen mentalitet enn ingeniørvitenskap.

3 Løsningsstrategi

Arbeidet med dette språket bør starte der det kan gi verdi *først*. Beregningsdokumenter er en god kandidat. Et godt beregningsdokument bør la oss uttrykke tekst og matematikk, og gi umiddelbar tilbakemelding. Mathcad

kommer meget nær, uten å treffe helt: (a) for tungvindt å programmere, (b) for tungvidt å skrive tekst og (c) for liten grad av kontroll over eksport.

Senere utvides økosystemet rundt. FEM-modeller bør hente parametre og konstanter fra beregningsdokumentet. Når vi selv regner ut parameterene vi trenger, skal vi ikke måtte gjøre dette for hånd og manuelt sørge for at modellen er oppdatert hver gang forutsetningene endres.

Vi går altså over til å modellere prosess, og ikke lenger bare et spesialtilpasset tilfelle av prosessen. Vi modellerer hva vi tenker, og ikke et gitt resultat av tankeprosessen. Vi kan lage strukturen vår selv. Tilsvarende utvikling har vært sentral innen informasjonsteknologi fra 50-tallet fram til i dag under paraplyer som *Structured programming* og *Literate programming*.

Hvorfor ikke bare eksisterende verktøy for programmering, for eksempel Python? Ingeniørproblemer er en helt annen klasse enn generelle programmeringsutfordringer. At vi ikke trenger å ta stilling til hvordan operativsystemer skal skrives, en webserver skal kommunisere med omverdenen eller hvordan en U-båt skal styres gjør at vi kan lage et system som er lettere å lære og gir bedre informasjon underveis. Løsningen er ikke Python-programmering, men en arbeidsprosess inspirert av programmering som passer til tankeflyten for å løse ingeniøroppgaver.

Konkret start:

1. Lag et system for levende beregningsdokumenter
2. La studenter bruke systemet for å lære mekanikk
3. Lær av erfaringer

4 Visjon og effekter

Hvor tungvindt er det i dag å undersøke effekten av variasjon av stivhet, opprissing og laster? Å sammenlikne to prosjekter? For ti binære valg har vi 2^{10}=1024 utfall. Å undersøke årsaksammenheng for dette problemet er praktisk umulig i dag.

Premisset om at det er utfallet til et enkelttilfelle stemmer labert i industri. I forskning stemmer det enda dårligere. Hvor stor del av tiden bruker master- og doktorgradsstudenter på å lære finurligheter i Abaqus? Å gjøre samme prosedyren gjentatt så de får resultater som faktisk kan sammenliknes? Hvor mye lenger tid bruker de på dette enn nødvendig? Hvor mye lenger kunne vi kommet med en raskere læringsprosess? Dagens verktøy er ikke skrevet for parameterstudier. Det er ikke noe vi trenger å akseptere.

I elementanalyse og programmering i dag er effekter unødvendig usynlige. Tiden fra du gjør noe til du får se en respons er mye lenger enn den trenger å være. Hva vil effekten for læring være dersom vi kan slippe elever løs og *prøve* konseptene de skal lære selv i noen minutter? Teori er i aller høyeste grad nødvendig, men vi bør vurdere *når* teorien skal innføres, og om den skal stå alene eller få noen eksempler. Hva om studenten direkte kan *prøve* forskjellige elementtyper og få se resultatene i sanntid? Hva om studenten kan få visualisert utnyttelse av moment- og skjærkapasitet samme øyeblikk som et nytt tverrsnitt velges? Hva om deformasjonsplottet direkte endres ved å gå fra en elastisk til en plastisk materialmodell? Ved å få bedre vektøy kan vi lære direkte av eksempler.

Noen prinsipper:

- Et beregningsdokument *er en brukerinteraksjon*. Det kan reagere på brukeren, oppmuntre til en dialog, heller enn monolog. Tilbakemelding skal gis *så snart som mulig*, slik at ny informasjon blir tilgjengelig umiddelbart.
- Resultater fra FEM-analyser skal vises i *sanntid*. Endringer av lasten skal umiddelbart gi effekt på momentdiagrammet.
- Organisering av informasjonen i beregningsprogrammer bør styres av *brukeren*, og ikke være begrenset til akkurat hva utvikleren har tenkt på.
- Førsteklasses støtte for matematikk og programmering er vitalt.
- Førsteklasses støtte for variasjon av inputparametre og sammenlikning av resultater som funksjon av parametervalg er vitalt.

5 Referanser

Relaterte domener:

- Matematikk
- Mekanikk
- Programmering
- Datavisualisering
- Brukerinteraksjon

Levende eksempler på prinsippene jeg argumenterer for å innføre for beregningsmekanikkl:

- Bret Victor: Inventing on Principle (54 minutter). Foredrag basert på prinsippet "Creators need an immediate connection to what they are creating", med prototyper som demonstrerer muligheter å få til dette
- Scratch er et levende eksempel på hvordan programmering *kan* være når vi ikke trenger å håndtere kompilatorteknikk og prosessorarkitektur
- OpenSCAD demonstrerer liveprogrammering av 3D-modeller. Hvorfor fungerer ikke dagens beregningsprogramvare slik?
- Elm er en god måte å lage brukerinteraksjoner. Elm er en levende demonstrasjon på to viktige prinsipper:
  - Tilbakemelding *under utvikling* er svært viktig. Det holder ikke at ting fungerer når alt er ferdig og alle feil er rettet, verktøyet må fungere så godt som mulig til utforskning og utvikling.
  - Et doktorgradsprosjekt som i dag preger verden
- Pollen brukes til å skrive bøker ved å se på en bok som et program. Resultatet av beregningen er en bok, i formater som PDF, HTML eller MOBI
- Jupyter Notebooks demonstrerer muligheten til å programmere med fokus på *data* i programmet i stedet for en abstrakt tolkning av kildekoden, samt muligheten til å kjøre én komponent av programmet av gangen