IN-KJM1900 — Forelesning 1

Simen Kvaal

Onsdag 25/10/2017

Oversikt

Litt om meg

- Jeg heter Simen Kvaal, `simen.kvaal@kjemi.uio.no`
- Jobber ved Hylleraas-senteret, tidligere CTCC (senter for fremragende forskning på kjemi)
- Teoretisk og beregningsbasert kjemi
- Om dere ønsker å treffe meg, avtal møte på forhånd

Prosjektoppgave

- Vi skal løse en prosjektoppgave om sur nedbør
- Stort miljøproblem på 70-80-tallet
- Forurensing fra Tyskland, Storbritannia, ...
- Fiskebestand nesten borte fra mange vassdrag i Sør-Norge, 36.000 km^2 (!)
- I dag et mindre problem.
- Spørsmål: hva er årsakssammenhengene mellom sur nedbør og forsuring av vann og jord?
- Vi skal gjenskape forskning fra 70-tallet i en prosjektoppgave!

Opplegg

- Forelesning 1 gang/uke (onsdager 10.15–12.00, 5 uker)
- Peer Instruction med Pingo.
- Gruppetimer 1 gang/uke (som før, 5 uker)
- Samretting (rom Ø186, onsdager 16.15–18.00)
- Prosjektoppgave: del I (2 uker), del II (3 uker)
- Begge må godkjennes

Oversikt

Birkenesmodellen

- Numerisk modell for vannbevegelse og kjemi i jordvæske
- Tilpasset observasjoner i nedbørsfelt i Birkenes kommune, Aust-Agder
- Flere publikasjoner på 70-90-tallet
- Kjemisk modell koblet til hydrologisk modell

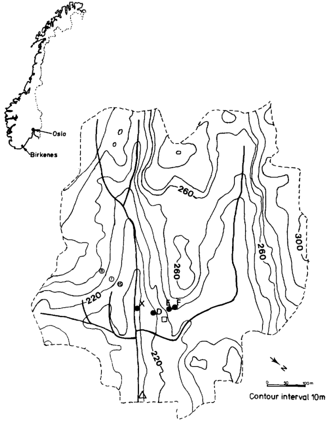
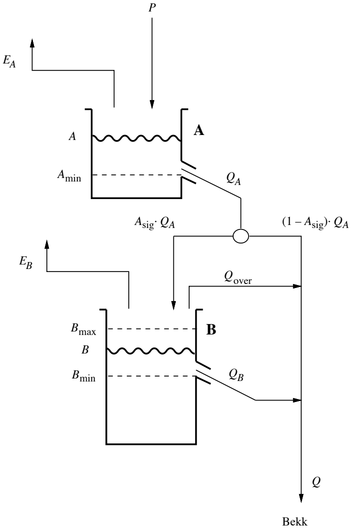


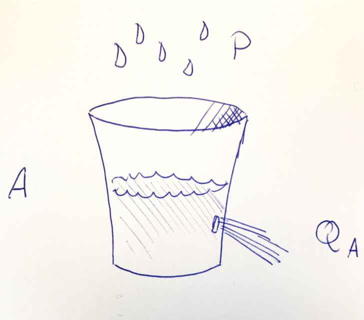
Fig. 1. The Birkenes catchment showing lysimeter plots (solid circles), piezometers (open circles), the weir in the main brook (triangle), and the pits where soil samples were taken (square).

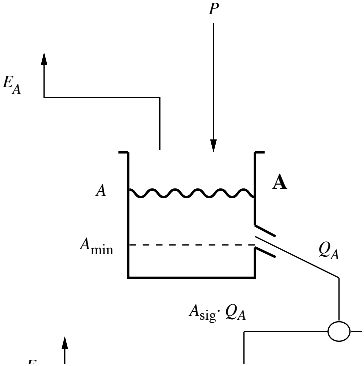
Hydrologisk delmodell: Kjapp intro

- Hydrologi: læren om vannet på jorda, dets kretsløp, etc.
- Her: transport av jordvæske i nedbørsfelt
- Nedbørsfelt modellert som reservoarer **A** og **B**
- Reservoar: vannstand, strøm inn og ut. Sammenkobling via strømmer.
- Avrenning til bekk. Fysiske målinger gjort her.



Det kan være nyttig å tenke på et reservoar som en bøtte med vann med et hull.





Kjemisk delmodell: kjapp intro

- Kjemisk delmodell: Ideell blanding av ioner løst i væsken. Ioner er kjemiske forbindelser med ladning.
- Sur nedbør som fortynnet svovelsyre H_2SO_4 i nedbøren P,

$$H_2SO_4 \longrightarrow 2H^+ + {SO_4}^{2-}$$

- Sulfat kommer med nedbøren: ${SO_4}^{2-}$ følger vannet gjennom modellen.
- Reaksjoner med jord: utlekking av giftig aluminium, etc.
- Vi følger konsentrasjonene av ioner i væsken

$$[H^+], [{SO_4}^{2-}], [Al^{3+}], [{HCO_3}^-], [Ca^{2+}]$$

$$[X] = \frac{\text{masse av } X}{\text{volum}}$$

- Ender opp med konsentrasjoner i bekken Q
- Miljøpåvirkning!

Birkenesmodellen sydd sammen

1. Først løser vi hydrologisk delmodell:

$$\text{vannivå} \quad A(t), \qquad B(t)$$

$$\text{strøm i bekken} \quad Q(t)$$

2. Så finner vi massene av ${SO_4}^{2-}$ i reservoarene

$$M_A(t), \qquad M_B(t)$$

   Konsentrasjoner:

$$C_A(t) = \frac{M_A(t)}{A(t)}, \qquad C_B(t) = \frac{M_B(t)}{B(t)}, \qquad C_Q(t)$$

3. De andre konsentrasjonene fra likevektsbetingelser + elektronøytralitet

Delprosjekt I

Utvikle verktøy, Python-modul: 2 uker

1. Innlesing av datafil med målinger av temperatur $T(t)$, nedbør $P(t)$ og $[SO_4{}^{2-}]$, observert strøm $Q_{\text{obs}}(t)$ i bekk
2. Newtons metode for ikkelineære likninger

   $$\text{finn } x \text{ slik at } f(x) = 0.$$

   Brukes i kjemisk delmodell.
3. Eulers metode for difflikninger

   $$\frac{d}{dt}y(t) = f(y(t), t)$$

   Brukes både i hydrologisk og kjemisk delmodell.

Delprosjekt II

Utvikle simuleringsprogram for birkenesmodell: 3 uker

- Gjenbruke Python-modul

1. Løsning av hydrologisk delmodell
2. Løsning av kjemisk delmodell
3. Beregning av konsentrasjoner

Oversikt

Moduler

- En modul er en samling med definisjoner i en `.py-fil`.
- Denne kan importeres i et skript slik at definisjonene blir tilgjengelige
- Til nå har dere kanskje mest brukt moduler, ikke skrevet de:

```
import numpy
x = numpy.linspace(0, 1, 500)
```

- Men det er veldig lett å lage sin egen modul! Du trenger kun å putte definisjoner i en egen Python-fil, så importere med `import`.

Moduler

`minmodul.py`

```
svaret = 42

def minfunksjon():
  print("Denne funksjonen er importert.")
```

Så kan vi bruke modulen:

```
import minmodul # leser minmodul.py

print(minmodul.svaret) # variabel definert i minmodul.py
minmodul.minfunksjon() # funksjon definert i minmodul.py
```

Moduler er nyttige!

- Moduler er kjempenyttige!
- Mange oppgaver gjøres mange ganger, og i ulike sammenhenger
- Løse oppgaver en gang for alle!
- Hjelper også tankene i en kompleks oppgave: skille uavhengige problemdeler fra hverandre

Tittel

Liveprogrammeringseksempel.

Oversikt

Hvorfor testfunksjoner?

- Et komplett program består av mange underprogrammer (funksjoner, klasser, $\cdots$) i samspill
- For at koden skal være robust, må den testes
- La oss si at funksjon B kaller funksjon A. Ett år senere redigerer du funksjon A og endrer litt på formatet på parametrene. Da ødelegges funksjon B, og du får bugs
- Testfunksjoner sørger for at du fanger opp slike problem

Pytest

- Skriv testfunksjoner som begynner på `test_` i kildefiler som begynner på `test_`.
- Ingen parametre. Ingen output. Kun `assert`-utsagn (kaster `AssertionError`-exceptions.)
- Du kaller ikke funksjonene selv.
- Kjør `pytest` fra kommandolinja. Pytest finner alle filene som starter på `test_` og alle funksjonene med navn som starter på `test_`, og kjører alle testene.
- Du får en rapport.

Testfunksjoner: generell form

```python
# Ingen argumenter, ingen returverdier
def test_funksjon():

    # Check that some success critera are met:
    if some_criteria_that_must_be_valid_if_code_is_sane:
        success = True
    else:
        success = False


    msg = "Message shown if test fails."

    # Check that success == True, and if not throw
    # an AssertionError with message msg

    assert success, msg
```

Live eksempel: `temperatures.py`

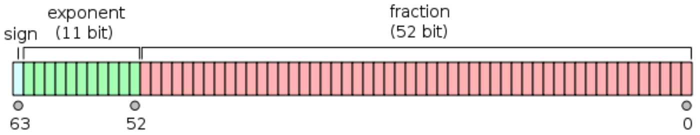
Dette eksempelet er enkelt, men relevant! Feilaktig bruk av enheter fikk Mars Climate Orbiter til å krasje i 1999.

Oversikt

Flytall: en orientering

- Når vi gjør beregninger på datamaskiner er de sjelden eksakte
  - ~~Modellfeil: Modellen er ikke riktig (eks. værmelding)~~
  - Regnefeil: Likningene løses kun timærmingsvis
- En datamaskin rekner med flyttall
- Enkelt sagt, *endelige desimalekspansjoner*,

$$y = \pm a \cdot 10^n, \quad a,\ n \text{ heltall}$$



- Ca. 15–17 gjeldende siffer (Python `float`, C `double`)
- Kun et endelig antall desimaler tilgjengelig: Representasjonsfeil

```
>>> from math import pi
>>> print pi
3.14159265359
```

Tap av signifikans og avrundingsfeil

- Alle beregninger har en endelig presisjon – avrundingsfeil
- Vi kan få fenomener som tap av signifikans.
- Rekkefølgen på operasjoner kan spille en rolle:

```
>>> 1 + 1e-100 - 1
0.0
>>> 1 - 1 + 1e-100
1e-100
>>>
```

- Eksakt svar i neste eksempel er 10^{-13} =`1e-13`:

```
>>> 1.0000000000000000 - 0.9999999999999
1.000310945187266e-13
```

Eksempel: numerisk derivasjon

- Den deriverte til $f(x)$ er

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Eksempel:

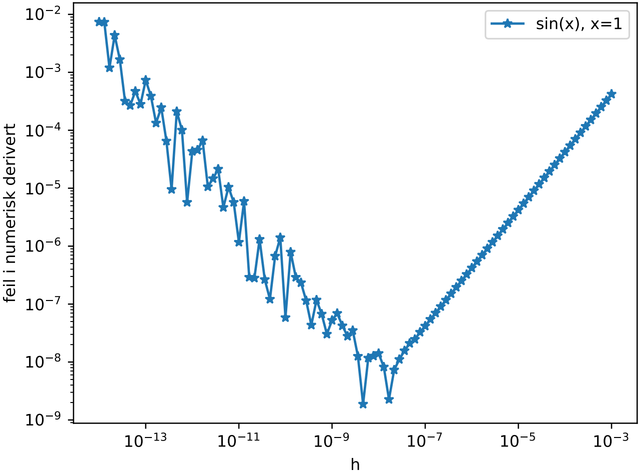
$$f(x) = \sin(x), \qquad f'(x) = \cos(x).$$

- I datamaskinen kan vi aldri la $h \longrightarrow 0$!
- Approksimasjon: velg en liten h, og regn ut

$$f'_h(x) = \frac{f(x+h) - f(x)}{h}$$

Eksempel: $f(x) = \sin(x)$

Vi kan ikke velge *h* så liten vi vil!



Oversikt

Likninger med én ukjent

- La $f(x)$ være en funksjon, for eksempel

$$f(x) = x^3 - 1.$$

- Betrakt problemet: finn x_* slik at

$$f(x_*) = 0$$

- En slik x_* kalles en "rot" av funksjonen $f(x)$
- For funksjonen $f(x) = x^3 - 1$ kan vi finne løsningen $x_* = 1$ ganske greit, men hva med dette eksempelet?

$$f(x) = x^5 - 4\cos(x)e^{\tan(x/\pi)} - 7$$

- Vi trenger approksimative metoder

Newtons metode

- Iterativ metode:
  1. Start med et gjett x_0 for x_*
  2. Gjenta en formel mange ganger:

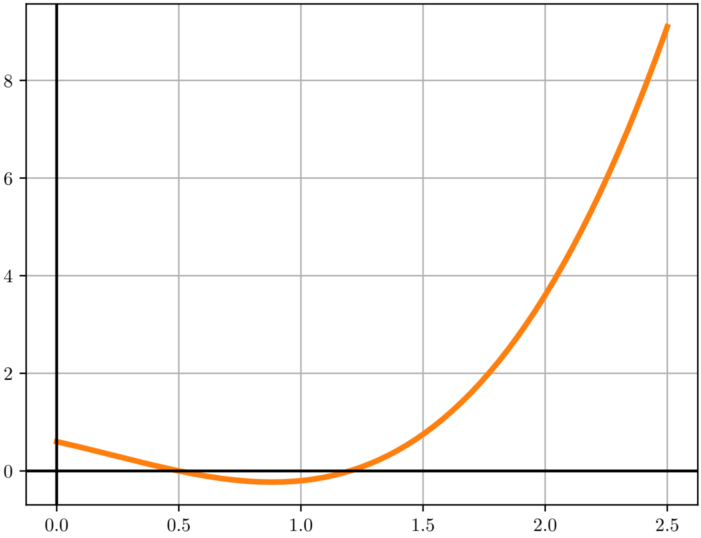
$$x_0 \longrightarrow x_1 \longrightarrow x_2 \longrightarrow \cdots$$

  3. Stopp når x_k har konvergert
- Formelen for Newtons metode er:

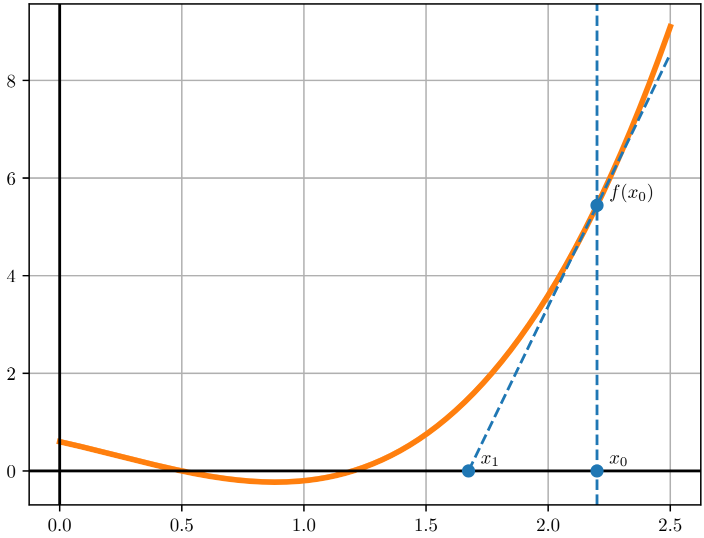
$$x_{k+1} = x_k - f'(x_k)^{-1} f(x_k).$$

- Vi må ha et uttrykk for både $f(x)$ og $f'(x)$.

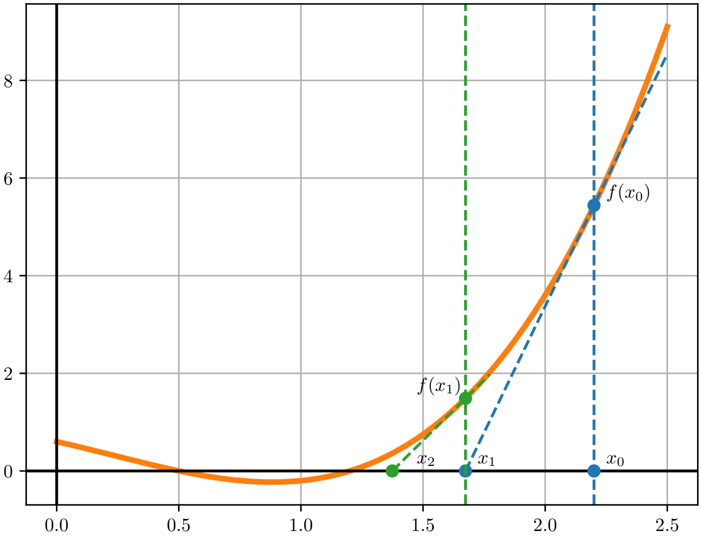
Illustrasjon: $f(x) = x^3 - 0.7x^2 - 1.1x + 0.6$



Illustrasjon: $f(x) = x^3 - 0.7x^2 - 1.1x + 0.6$



Illustrasjon: $f(x) = x^3 - 0.7x^2 - 1.1x + 0.6$



Eksempel

- $f(x) = (x+1)(x-0.5)(x-1.2) = x^3 - 0.7x^2 - 1.1x + 0.6$
- Konvergenshistorikk:

```
x0 = 2.200000
At iteration 0, x = 1.6738878143133462.
At iteration 1, x = 1.3741331494162794.
At iteration 2, x = 1.2372954723615801.
At iteration 3, x = 1.2023502104483390.
At iteration 4, x = 1.2000103267171331.
At iteration 5, x = 1.2000000002008111.
At iteration 6, x = 1.2000000000000000.
```

- Når vi nærmer oss x_*: Antall korrekte siffer dobles per iterasjon
- "Kvadratisk konvergens"
- Hvor ville du startet dersom du ønsket roten $x_* = 1.0$?

Nytt eksempel (litt for moro skyld)

- Newtons metode fungerer også på komplekse polynom
- Komplekse tall har en real og imaginær del:

$$x = a + ib, \quad i^2 = -1$$

$$x + x' = (a + a') + i(b + b'), \quad xx' = (aa' - bb') + i(ab' + b'a)$$

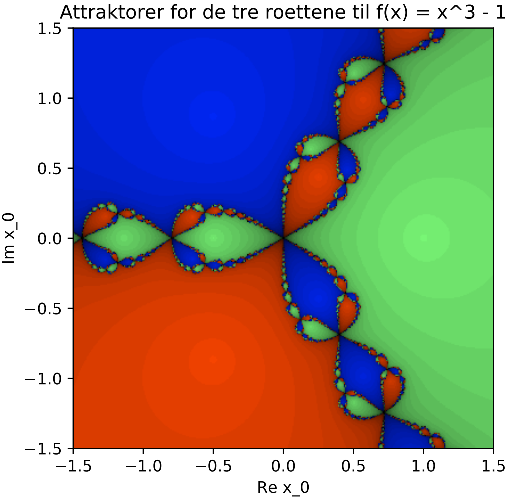
- Så komplekse tall er som vektorer i planet der vi også har multiplikasjon definert (Caspar Wessel, 1745–1818)
- x_0 er her kompleks, og vi kan få ulik x_* avhengig av denne.

$$f(x) = x^3 - 1$$

$$x_* \text{ kan være} \quad 1, \quad \tfrac{1}{2} + i\tfrac{\sqrt{3}}{2}, \quad \tfrac{1}{2} - i\tfrac{\sqrt{3}}{2}$$

- Dette er også et eksempel på at moduler er nyttige! Vi bruker samme implementasjon av Newtons metode som tidligere

Resultat


Attraktorer for de tre roettene til f(x) = x^3 - 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Im x_0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Re x_0

Oversikt

Hva er en difflikning?

- En liking der den ukjente er en (eller flere) funksjoner
- Vi er interessert i initialverdiproblemer: Finn $y(t)$ slik at

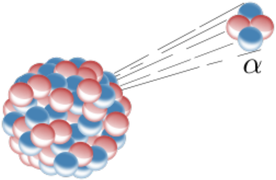
$$\frac{d}{dt}y(t) = f(y(t), t), \qquad y(0) = a$$

  der $f(y, t)$ er en kjent funksjon
- Likingene for vannbevegelse er på denne formen
- Typisk: vi må løse likningene *numerisk*

Eksempel: radioaktivt henfall

- Radioaktive kjerner kan henfalle ved å sende ut en ${}^4_4$He-kjerne (α-partikkel)
- Over en tid δt sannsynlighet $\lambda \delta t$ for at en kjerne henfaller. Halveringstid,

$$t_{1/2} = \lambda \ln(2)$$



- I snitt vil $N\lambda\delta t$ kjerner henfalle, N=antall kjerner i en prøve
- Masseregnskap:

$$\delta N = -\lambda N \delta t$$

- Difflikning: $\delta t \longrightarrow 0$,

$$\frac{dN}{dt} = -\lambda N, \quad N(0) = N_0$$

Eulers metode

$$\frac{d}{dt}y(t) = f(y(t), t), \quad y(0) = a$$

- De fleste difflikninger må løses numerisk
- Eulers metode er kanskje den enkleste numeriske metoden
- Basert på definisjonen av den deriverte:

$$\frac{d}{dt}y(t) = \lim_{\delta t \to 0} \frac{1}{\delta t}[y(t + \delta t) - y(t)]$$

- Anta at vi kjenner $y(t)$, velger fiksert $\delta t > 0$ og løser for $y(t + \delta t)$:

$$y(t + \delta t) = y(t) + \delta t \cdot f(y(t), t).$$

Eulers metode II

Input:

- Steglengde h og ønsket tidsintervall $0 \leq t \leq t_{\text{slutt}}$
- Funksjon $f(y, t)$
- Initialverdi $y(0) = a$

Output:

- Vektor $y = (y_0, y_1, \cdots, y_k, \cdots)$ med beregnede approksimasjoner til $y(t_k)$, $t_k = kh$.

Algoritme:

1. Velg tidssteg $h > 0$. Sett $y_0 = a$. Sett $k = 0$.
2. Sett
$$y_{k+1} = y_k + h \cdot f(y_k, t_k), \quad t_k = kh.$$
3. Dersom $t_k < t_{\text{slutt}}$, gå til 2

Radioaktivt henfall og Eulers metode

Live-programmering av

$$N_{k+1} = N_k - \lambda h \cdot N_k$$

$$N_0 = 100$$

$$\lambda = 0.5 \ln(2)$$

$$h = 1$$

Plotting av resultat