Forelesningsnotater, FYS1100 uke 5

Joakim Bergli

1 Numerisk løsning av difflikninger

Forrige uke lærte vi hvordan vi kan løse noen differensiallikninger (de som er separable) eksakt analytisk. For eksempel så vi på et legeme som faller fritt, der vi tar hensyn til luftmotstanden. For å gjøre det enkelt å regne valgte vi å anta at luftmotstanden er proporsjonal med farta, slik at $F_L = kv$. Vi har da at det er to krefter, tyngden og luftmotstanden, og om vi velger positiv retning nedover (slik at farta er positiv) sier Newtons 2. lov at

$$ma = mg - kv.$$

Når vi så bruker at $a = \frac{dv}{dt}$ får vi differensiallikninga

$$\frac{dv}{dt} = g - \frac{k}{m}v. \tag{1}$$

Vi løste den og fant at

$$v(t) = \frac{m}{k}(g - Ae^{-\frac{k}{m}t})$$

for en positiv konstant A. Denne kosntanten bestemmers av initialbetinglesen, og hvis vi starter i ro, $v(0) = 0$ får vi $A = g$ og

$$v(t) = \frac{mg}{k}(1 - e^{-\frac{k}{m}t}).$$

For mange likninger er det umlig å finne en eksakt analytisk løsning, og vi bruker isteden en numerisk metode. Det vil si at vi får en datamaskin til å regne ut tilnærmede verdier for funksjonen.[1] Vi vil vise hvordan vi kan gjøre dette med den kanskje enkleste og mest intuitive metoden (Eulers metode) for likninga (1).

Vi husker definisjonen akselerasjon som den av den deriverte av farta

$$a(t) = \frac{dv}{dt} = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t}.$$

Istedenfor å ta grensen $\Delta t \to 0$ vil vi nå la datamaskina jobbe med en liten men endelig Δt. Det betyr at vi regner ut løsninga bare på et diskret sett av tidspunkter $t_0, t_1, t_2, \ldots$, med $t_{i+1} - t_i = \Delta t$ og tilnærmer

$$\frac{dv}{dt} \approx \frac{v_{i+1} - v_i}{\Delta t}$$

[1] Vi kan som oftest velge hvor nøyaktig vi vil ha verdiene, men det tar lengre tid å beregne med større nøyaktighet, så vi må gjerne velge en passe balanse mellom nøyaktig og raskt. Det finnes mange forskjellige måter å gjøre slike beregninger, og de har ulike styrker og svakheter avhengig av hva slags likning man vil løse. Vi kommer mer tilbake til dette seinere (uke 10).

der $v_i = v(t_i)$. Da har vi at

$$\frac{v_{i+1} - v_i}{\Delta t} \approx a(t)$$

som vi kan omforme til

$$v_{i+1} = v_i + a(t)\Delta t. \tag{2}$$

Hvis vi derfor starter med initialfarta $v_0 = v(t_0)$ kan vi regne ut farta v_1 ved tida t_1 ved å bruke likning (2). Når vi har den bruker vi samme likning (men med oppdatert akselerasjon) til å finne v_2 ved tida t_2, og slik fortsetter vi så langt vi har behov for. Denne prosessen hjelper oss også å forstå betydningen av initialbetingelsen i en differensiallikning. Likninga (1) gir oss en regel for hvordan funksjonen $v(t)$ endrer seg (som er det den deriverte forteller oss). Men for å vite hvordan funksjonen er trenger vi ikke bare å vite hvordan den endrer seg, men også hvor den starter. I likning (2) må vi altså ha en startverdi v_0, men kan så regne ut alle seinere v_i.

Likning (1) gir oss akselerasjonen som en funksjon av farta v

$$a(v) = g - \frac{k}{m}v$$

og vi kan lure på hvilken fart vi skal velge når vi skal renge ut a i (2). Det enkleste er å velge v_i siden vi allerede kjenner denne når vi skal regne ut v_{i+1}. Fysisk svarer dette til at vi tenker at akselerasjonen er konstant lik $a(v_i)$ i hele det lille tidsintervallet $[t_i, t_i + \Delta t]$. For å regne farta ved alle tidene t_i må vi altså bruke likninga

$$v_{i+1} = v_i + a(v_i)\Delta t \tag{3}$$

om og om igjen.

2 Eksempel: Fritt fall med luftmotstand

La oss skrive et program som gjør stegene over for eksempelet med fritt fall med luftmotstand.

```
import numpy as np
import matplotlib.pyplot as plt

# Definerer konstanter
g = 9.81
k = 1.0
m = 1.0

# Definerer initalverdien
v0 = 0.

# Definerer tidsintervall og noen konstanter
tmax = 10.0
dt = 0.01
c = k/m
n = int(tmax/dt)

# Lager arrays for å lagre resultater og setter startverdiene
a = np.zeros(n)
v = np.zeros(n)
t = np.zeros(n)
v[0] = v0
t[0] = 0
```

```
# Integrasjonsløkka
i = 0
while i<n-1:
    a[i] = g - c*v[i]
    v[i+1] = v[i]+a[i]*dt
    t[i+1] = t[i]+dt
    i = i+1

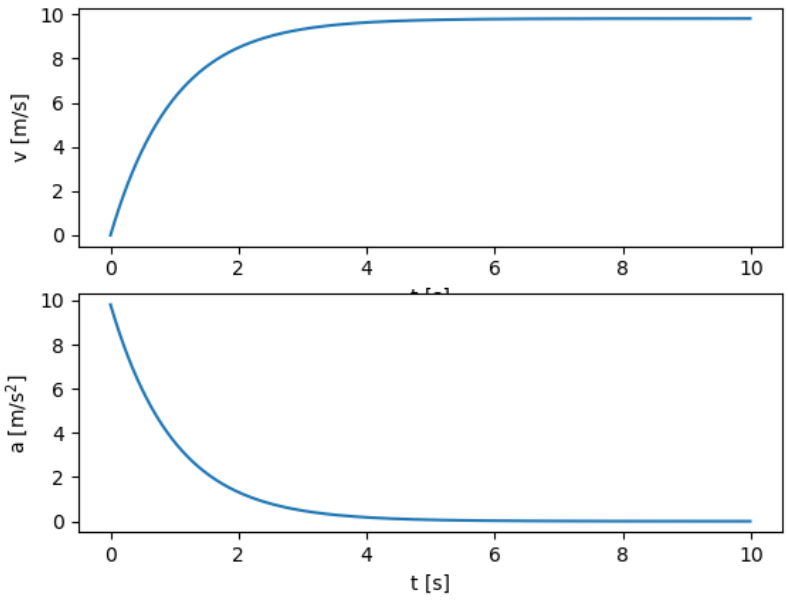

# Plotter resultatene

plt.subplot(2,1,1)
plt.plot(t,v)
plt.xlabel('t [s]')
plt.ylabel('v [m/s]')

plt.subplot(2,1,2)
plt.plot(t,a)
plt.xlabel('t [s]')
plt.ylabel('a [m/s$^2$]')

plt.savefig('fallMedLuftmotstand.png')
plt.show()
```

Programmet plotter både fart og akselerasjon som funksjoner av tida



Som vi forventer går akselerasjonen mot null ettersom luftmotstanden nærmer seg tyngden, og farta øker og nærmer seg terminalfarta. Så langt har vi bare fått de samme resultatene som vi fikk fra den analytiske løsninga. Men husk at vi valgte luftmotstand som er proporsjonal med farta. Dette var ikke fordi det var realistisk for de fleste "vanlige" gjenstander (baler, steiner...) som faller i luft, men fordi det gav enklere regning. Mer realistisk er at luftmotstanden er proporsjonal med kvadratet av farta, slik at $F_L = kv^2$. Difflikninga vi må løse blir da

$$\frac{dv}{dt} = g - \frac{k}{m}v^2.$$

Denne er også separabel, og vi kan løse den på samme måte som likning (1), men det er litt vanskeligere (prøv det selv om du vil ha litt ekstra trening!). I programmet vårt er det imidlertid veldig lett å bytte til denne mer realistiske situasjonen. Vi bytter bare ut linja

```
a[i] = g - c*v[i]
```

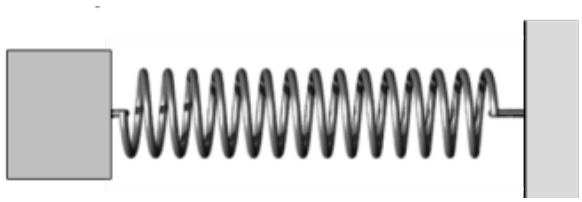
med

```
a[i] = g - c*v[i]**2
```

og kjører programmet. Enda mer realistisk ville det være å ha en funksjon $F_L = f(v)$ som er lineær for små v og så går over til kvadratisk for store v. Kanskje vi til og med har målt denne funksjonen for det legemet vi bruker, og den er verken helt lineær eller helt kvadratisk noe sted. Da er det nok små sjanser for at vi kan løse likninga analytisk. Men i programmet trenger vi bare å sette inn den riktige funksjonen i den ene kodelinja, og det vil kunne regne ut hvordan farta øker i det tilfellet.

3 Hvis krafta avhenger av posisjonen

Eksempelet med fritt fall med luftmotstand er ekstra enkelt fordi både tyngden og luftmotstanden er uavhengige av posisjonen. Mange problemer i mekanikk har en kraft som avhenger av posisjonen. Noe av det enkleste vi kan analysere er en kloss med massen m festet til ei fjær med fjærkonstant k. Vi tenker at klossen kan bevege seg horisontalt uten friksjon, slik at den eneste krafta som påvirker den horisontale bevegelsen er fjærkrafta.



Om vi lar klossen ha posisjonen x og origo være i likevektsposisjonen til klossen (der fjæra verken er strukket eller presset sammen) er fjærkrafta $F = -kx$. Her er det negativt fortegn fordi krafta vil virke i motsatt retning av den retningen klossen har beveget seg til. Newtons 2. lov er da

$$ma = -kx,$$

som blir til differensiallikninga

$$m\frac{d^2x}{dt^2} = -kx \tag{4}$$

når vi husker at akselerasjonen er den andrederiverte av posisjonen med hensyn på tid. Vi kaller dette en andreordens differensiallikning siden den inneholder en andrederivert. Vi skal seinere lære hvordan vi kan løse denne likninga analytisk, men det er lett å gjøre det numerisk. Vi bruker samme framgangsmåten som før, bare at vi må vite både farta og posisjonen ved starttidspunktet. Formelt er det vi gjør å dele den andreordens likninga i to førsteordens ved å innføre farta $v(t)$ som en ukjent funksjon i tillegg til posisjonen. Istendefor likning (4) har vi da de to likningene

$$\frac{dv}{dt} = a(x,v) = -\frac{k}{m}x.$$
$$\frac{dx}{dt} = v.$$

Istedenfor likning (3) får vi da generelt

$$\begin{aligned} v_{i+1} &= v_i + a(x_i, v_i)\Delta t \\ x_{i+1} &= x_i + v_{i+1}\Delta t. \end{aligned} \tag{5}$$

Fysisk svarer dette til at vi tenker at akselerasjonen er $a(x_i, v_i)$ i hele intervallet $[t_i, t_i + \Delta t]$ og regner vi ut farta v_{i+1}. Så tenker vi at farta er v_{i+1} i hele intervallet $[t_i, t_i + \Delta t]$ og regner ut x_{i+1}. Vi kan lure på hvorfor vi bruker farta v_{i+1} som skal gjelde i slutten av intervallet for hele intervallet og ikke farta i starten, v_i. Eller vi kunne tenke at det ville bli bedre om vi valgte gjennomsnittsfarta $\frac{1}{2}(v_i + v_{i+1})$. Det finnes mange slike valg, og de er gode for ulike problemer. Den metoden som er gitt i likning (5) kalles Euler-Cromer metoden, og man kan vise at den er god for mange problemer med ting som svinger periodisk. Vi skal komme litt tilbake til dette seinere (når vi har lært om Taylorrekker). I FYS1100 kan dere alltid bruke denne metoden, og det vil gi gode resultater.

Her er et program som implementerer Euler-Cromer metoden for likning (4).

```
# Euler-Cromers metode for harmonisk oscillator

import numpy as np
import matplotlib.pyplot as plt

# Definerer parametre, startposisjon og startfart.
m = 1
k = 1
x0 = 2
v0 = 0

# Definerer hjelpevariabel (c), tidssteg og makstid
c = k/m
dt = 0.01
tMax = 50

# a,v,x,t er akselerasjon, fart, posisjon og tid
# Lager arrayer til å lagre resultatene

n = int(tMax/dt)
t = np.zeros(n)
a = np.zeros(n)
v = np.zeros(n)
x = np.zeros(n)

# Setter initialverdiene i starten av arrayene

x[0] = x0
v[0] = v0

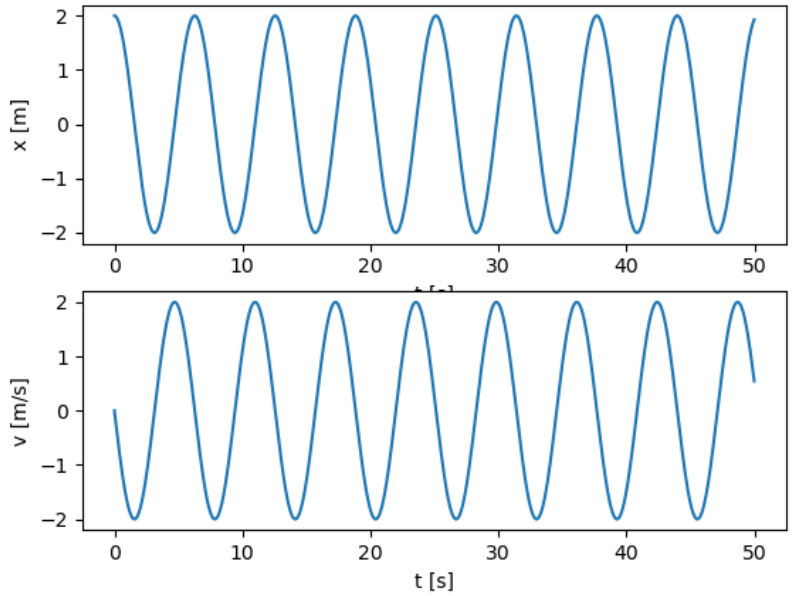
# Integrasjonsløkka der vi regner ut nye verdier for x og v fra de gamle
for i in range(n-1):
    t[i+1] = i*dt
    a[i] = -c*x[i]
    v[i+1] = v[i] + a[i]*dt
    x[i+1] = x[i] + v[i+1]*dt
```

```
# Plotter resultatene
plt.subplot(2,1,1)
plt.plot(t,x)
plt.xlabel('t [s]')
plt.ylabel('x [m]')

plt.subplot(2,1,2)
plt.plot(t,v)
plt.xlabel('t [s]')
plt.ylabel('v [m/s]')

plt.savefig('HO.png')
plt.show()
```

Resultatet blir



Prøv om du kan tolke de to grafene.