Forelesning IN1900 – 2 November 2023

Ole Christian Lingjærde
Institutt for informatikk, Universitetet i Oslo

Uke: 30 Oktober – 5 November, 2023

Dagens tema

Ordinære differensiallikninger (ODE'er)

Hvordan løse ODE'er - generell algoritme

Eksempler på løsing av ODE'er

Modulen ODESolver

Løsning av ordinære differensiallikninger

- ODE = Ordinary Differential Equation
- Likning hvor den ukjente er en funksjon $u(t)$
- Differential: knytter sammen $u(t)$, $u'(t)$ (og evt høyereordens deriverte)
- Ordinary: ser bare på deriverte i én variabel (f.eks. t)

Vi kommer til å se på to varianter av ODE'er:

- Skalar ODE: en likning
- Vektor ODE: flere likninger (likningssystem)

Å løse differensiallikninger
analytisk (= matematisk)
er ofte veldig vanskelig

Eksempel A

Anta at vi skal finne funksjonen $u(t)$ når vi vet at

$$u'(t) = t^3$$

Vi kan integrere på begge sider:

$$u(t) = \frac{1}{4}t^4 + C$$

Hvis vi i tillegg har en *initialbetingelse* $u(0) = 1$ kan vi finne C:

$$u(t) = \frac{1}{4}t^4 + 1$$

Eksempel B

Anta at vi skal finne funksjonen $u(t)$ når vi vet at

$$u'(t) = f(t)$$

Vi kan igjen integrere begge sider:

$$u(t) = \int f(t)\,dt + C$$

Har vi en initialbetingelse $u(0) = u_0$ kan vi igjen finne C.

Vi har sett tidligere i kurset hvordan man kan regne ut integraler numerisk.

Eksempel C

Eksempel A og B var veldig enkle differensiallikninger, siden vi kunne finne $u(t)$ bare ved å integrere på begge sider. Vi ser nå på en likning hvor u(t) også forekommer i høyresiden:

Anta at vi skal finne funksjonen $u(t)$ når vi vet at

$$u'(t) = \alpha u(t)$$

Vi prøver som før å integrere på begge sider:

$$u(t) = \alpha \int u(t)\, dt + C$$

Vi har funnet et uttrykk for $u(t)$, men høyresiden inneholder den ukjente funksjonen!

Hvordan løse Eksempel C?

Eksempel C er såpass enkel at vi kan løse den matematisk:

Vi skal finne $u(t)$ når

$$u'(t) = \alpha u(t)$$

Vi flytter om:

$$\frac{u'(t)}{u(t)} = \alpha$$

Vi får en smart innsikt og ser at dette kan skrives

$$(\ln u(t))' = \alpha$$

Vi integrerer på begge sider:

$$(\ln u(t)) = \int \alpha \, dt = \alpha t$$

Vi tar $exp(.)$ på begge sider:

$$u(t) = e^{\alpha t}$$

Kan vi si noe om løsningen til en ODE (uten å løse den)?

Eksempel A

Anta at vi har likningen:

$$u'(t) = a$$

Hva vi kan lese ut av likningen:

Stigningstall a for alle verdier av t

Altså: løsningen må være en rett linje

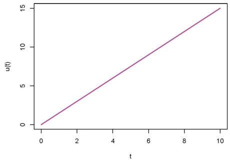
Eksempel A

Anta at vi har likningen:

$$u'(t) = a$$

Hva vi kan lese ut av likningen:

- Stigningstall a for alle verdier av t
- Altså: løsningen må være en rett linje



Eksempel B

Anta at vi har likningen:

$$u'(t) = a \cdot u(t) \qquad (a > 0)$$

Her kan vi lese ut av likningen:

> Stigningstallet er proporsjonalt med høyden til kurven
> Altså: en funksjon som vokser raskere og raskere

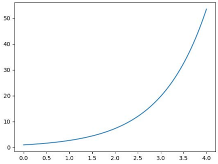
Eksempel B

Anta at vi har likningen:

$$u'(t) = a \cdot u(t) \qquad (a > 0)$$

Her kan vi lese ut av likningen:

- Stigningstallet er proporsjonalt med høyden til kurven
- Altså: en funksjon som vokser raskere og raskere



Eksempel C

Anta at vi har likningen:

$$u'(t) = u(t)(1 - u(t))$$

Her kan vi lese ut av likningen:

> Stigningstallet er 0 når $u(t) = 0$ eller $u(t) = 1$.
> Stigningstallet er positivt når $0 < u(t) < 1$

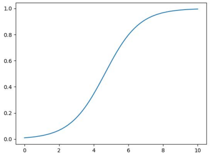
Eksempel C

Anta at vi har likningen:

$$u'(t) = u(t)(1 - u(t))$$

Her kan vi lese ut av likningen:

- Stigningstallet er 0 når $u(t) = 0$ eller $u(t) = 1$.
- Stigningstallet er positivt når $0 < u(t) < 1$



Eksempel D

Anta at vi har likningen:

$$u'(t) = 7u(t)^2 t^3$$

Anta $t > 0$:

- Stigningstallet er da ≥ 0
- Hvis $u(0) > 0$ stiger løsningen stadig raskere
- Hvis $u(0) < 0$ er det mer uklart. Forblir $u(t)$ negativ eller blir
- $u(t)$ positiv og stiger raskt?

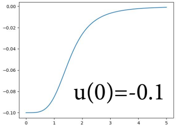
Eksempel D

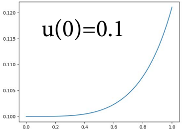
Anta at vi har likningen:

$$u'(t) = 7u(t)^2t^3$$

Anta $t > 0$:

- Stigningstallet er da ≥ 0
- Hvis $u(0) > 0$ stiger løsningen stadig raskere
- Hvis $u(0) < 0$ er det mer uklart. Forblir $u(t)$ negativ eller blir
- $u(t)$ positiv og stiger raskt?





Numerisk løsing av ODE'er

Å løse differensiallikninger analytisk (finne en formel) kan være svært komplisert eller umulig. I fortsettelsen ser vi på hvordan en i Python kan løse slike likninger numerisk og visualisere løsningen.

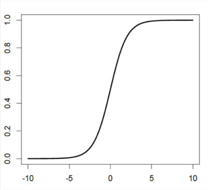
Likning

$$\boxed{x'(t) = x(t)\,(1-x(t))}$$

Numerisk løsning

t	x(t)
0.0	0.500
0.1	0.525
0.2	0.550
0.3	0.574
...	...
5.0	0.993

Grafisk løsning



Numerisk løsing av ODE'er: hovedideen

Hovedideen bak alle de metodene vi skal se på for å løse ODE'er numerisk er egentlig svært enkel og er basert på følgende resonnement:

> Vi starter med en ODE på formen $u'(t) = f(t, u(t))$
> Vi erstatter $u'(t)$ med $(u(t+h) - u(t))/h$
> Vi får da en *differenslikning* som vi enkelt kan løse med metoder vi har sett på tidligere i kurset hvis vi kjenner $u(0)$.

NB: det finnes diverse andre approksimasjoner til $u'(t)$ enn den over, og det gir ulike ODE-løsningsmetoder.

Numerisk løsing av ODE'er: hovedideen

Hovedideen bak alle de metodene vi skal se på for å løse ODE'er numerisk er egentlig svært enkel og er basert på følgende resonnement:

- Vi starter med en ODE på formen $u'(t) = f(\mathrm{t}, \mathrm{u}(\mathrm{t}))$
- Vi erstatter $u'(t)$ med $(u(t+h) - u(t))/h$
- Vi får da en *differenslikning* som vi enkelt kan løse med metoder vi har sett på tidligere i kurset hvis vi kjenner $u(0)$.

NB: det finnes diverse andre approksimasjoner til $u'(t)$ enn den over, og det gir ulike ODE-løsningsmetoder.

Numerisk løsningsmetode: trinn for trinn

ODE: $u'(t) = 3u(t)$

Trinn 1:

Vi setter inn formelen $u'(t) \approx (u(t+h) - u(t))/h$ i likningen:

$$\frac{u(t+h)-u(t)}{h} \approx 3u(t)$$

Trinn 2:

Vi rearrangerer formelen:

$$u(t+h) \approx u(t) + 3h \cdot u(t)$$

Trinn 3:

Hvis vi kjenner $u(0)$ kan vi nå enkelt finne $u(h)$, $u(2h)$, $u(3h)$, osv.

Løsningsalgoritme (skisse)

ODE: $u'(t) = 3u(t), \; u(0) = U_0$

Vi ønsker å finne løsningen $u(t)$ for $0 \leq t \leq T$.

1) Definer arrayer:

```
t = np.linspace(0, T, n+1)
u = np.zeros(n+1)
```

2) Sett inn initialbetingelsen:

```
u[0] = U0
```

3) Regn ut løsningen steg for steg (`h = T/n`):

```
u[1] = u[0] + 3*h*u[0]
u[2] = u[1] + 3*h*u[1]
u[3] = u[2] + 3*h*u[2]
```

....osv....

Implementasjon i Python

```
import numpy as np

# Sett konstanter
T = 2       # Løsningsintervall [0,T]
n = 50      # Antall løsningspunkter er n+1
U0 = 1      # Initialbetingelse

# Definer arrayer
t = np.linspace(0, T, n+1)
u = np.zeros(n+1)

# Beregn steglengde (h)
dt = T/n

# Sett inn initialbetingelsen
u[0] = U0

# Finn u[1], u[2], ...
for k in range(n):
    u[k+1] = u[k] + dt * 3 * u[k]
```

Resultat

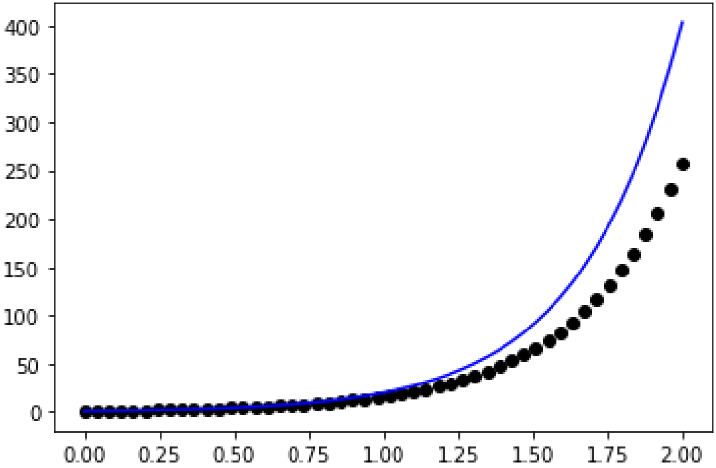
t	u
0.000	1.000000
0.041	1.120000
0.082	1.254400
0.122	1.404928
0.163	1.573519
0.204	1.762342
0.245	1.973823
0.286	2.210681
0.327	2.475963
....	....
....	....
1.918	205.706050
1.959	230.390776
2.000	258.037669

Implementasjon med plotting

Vi legger til følgende kode for å plotte den numeriske løsningen `u[0], u[1], ...` sammen med den eksakte løsningen $u(t) = e^{3t}$:

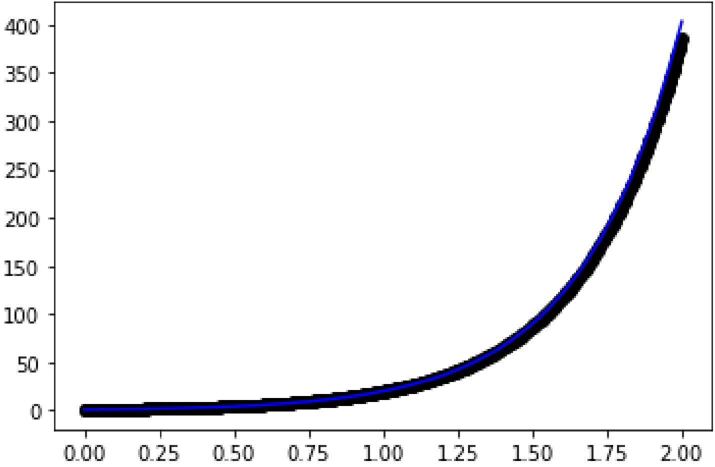
```
import matplotlib.pyplot as plt
plt.plot(t, u, 'ko')
plt.plot(t, np.exp(3*t), 'b-')
plt.show()
```

Resultat


400
350
300
250
200
150
100
50
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Flere løsningspunkter

Den numeriske løsningen på forrige slide avviker mer og mer fra den korrekte løsningen når t vokser. Vi kan få en mer nøyaktig løsning ved å øke antall løsningspunkter til $n = 500$:



Numerisk løsing av $u'(t) = u(t)(1 - u(t))$

ODE: $u'(t) = u(t)(1 - u(t)), \;\; u(0) = U_0$.

Trinn 1:

Vi setter inn formelen $u'(t) \approx (u(t+h) - u(t))/h$ i likningen:

$$\frac{u(t+h) - u(t)}{h} \approx u(t)(1 - u(t))$$

Trinn 2:

Vi rearrangerer formelen:

$$u(t+h) \approx u(t) + h \cdot u(t)(1 - u(t))$$

Trinn 3:

Vi bruker formelen over til å regne ut $u(h)$, $u(2h)$, $u(3h)$, osv.

Løsningsalgoritme (skisse)

Vi ønsker å finne løsningen $u(t)$ for $0 \leq t \leq T$.

1) Lag t-array og initier u-array:

```
t = np.linspace(0, T, n+1)
u = np.zeros(n+1)
```

2) Sett inn initialbetingelsen:

```
u[0] = U0
```

3) Regn ut løsningen steg for steg (`h = T/n`):

```
u[1] = u[0] + h * u[0] * (1-u[0])
u[2] = u[1] + h * u[1] * (1-u[1])
u[3] = u[2] + h * u[2] * (1-u[2])
```

....osv....

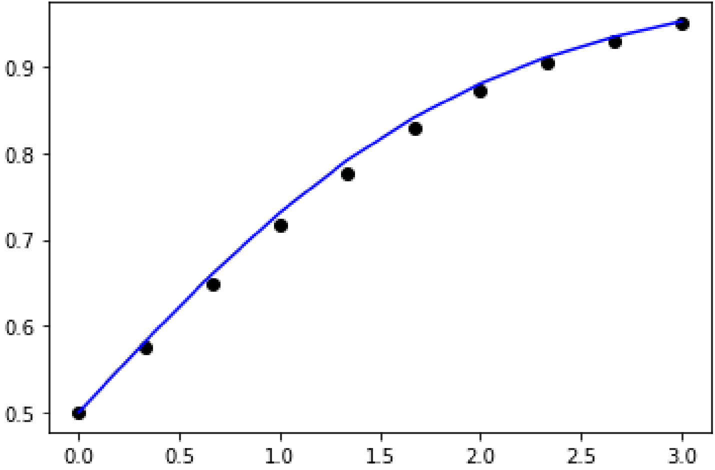
Implementasjon i Python

```
import numpy as np

# Sett konstanter
T = 3      # Løsningsintervall [0,T]
n = 10     # Antall løsningspunkter er n+1
U0 = 0.5   # Initialbetingelse

# Lag arrayer til å holde t- og u-verdier
t = np.linspace(0, T, n+1)
u = np.zeros(n+1)

# Beregn steglengde (h)
dt = T/n

# Sett inn initialbetingelsen
u[0] = U0

# Finn u[1], u[2], ...
for k in range(n):
    u[k+1] = u[k] + dt * u[k] * (1 - u[k])
```

Resultat

```
t          u
0.000      1.000000
0.041      1.000000
0.082      1.000000
0.122      1.000000
0.163      1.000000
0.204      1.000000
0.245      1.000000
0.286      1.000000
....    ....
....    ....
1.918      1.000000
1.959      1.000000
2.000      1.000000
```

Resultat



Sammenlikning av de to eksemplene

```python
import numpy as np

# Sett konstanter
T = 3      # Løsningsintervall [0,T]
n = 10     # Antall løsningspunkter er n+1
U0 = 0.5   # Initialbetingelse

# Lag arrayer til å holde t- og u-verdier
t = np.linspace(0, T, n+1)
u = np.zeros(n+1)

# Beregn steglengde (h)
dt = T/n

# Sett inn initialbetingelsen
u[0] = U0

# Finn u[1], u[2], ...
for k in range(n):
    u[k+1] = u[k] + dt * u[k] * (1 - u[k])    #### ENDRET
```

NB: det er bare én linje som måtte endres!

Generell metode for å løse ODE'er

Eksemplene ovenfor tyder på at vi kan lage ett Python-program som kan løse en hvilken som helst ODE:

Hvis likningen er $u'(t) = u(t)$ blir for-løkken:

```
for k in range(n):
    u[k+1] = u[k] + dt * u[k]
```

Hvis likningen er $u'(t) = u(t)(1 - u(t))$ blir for-løkken:

```
for k in range(n):
    u[k+1] = u[k] + dt * u[k] * (1-u[k])
```

Generelt: hvis likningen er $u'(t) = f(u(t), t)$ blir for-løkken:

```
for k in range(n):
    u[k+1] = u[k] + dt * f(u[k], t[k])
```

Forward Euler metoden (skisse)

Den generelle løsningsalgoritmen som vi nettopp har sett på kalles *Forward Euler* og kan skisseres slik:

```
import numpy as np

# Sett konstanter
T = ...       # Løsningsintervall [0,T]
N = ...       # Antall løsningspunkter
U0 = ...      # Initialbetingelse

# Lag arrayer til å holde t- og u-verdier
t = np.zeros(N+1)
u = np.zeros(n+1)

# Gi startverdier
u[0] = U0
t[0] = 0
dt = T/N

# Finn u[1], u[2], ...
for n in range(N):
   t[n+1]  = t[n] + dt
   u[n+1] = u[n] + dt * f(t[n], u[n])
```

Forward Euler implementert som funksjon

```
"""
Implementation of the ForwardEuler method as a function.
"""
import numpy as np
import matplotlib.pyplot as plt


def ForwardEuler(f, U0, T, N):
    """Solve u'=f(t,u), u(0)=U0, with n steps until t=T."""
    import numpy as np
    t = np.zeros(N+1)
    u = np.zeros(N+1)  # u[n] is the solution at time t[n]

    u[0] = U0
    t[0] = 0
    dt = T/N

    for n in range(N):
        t[n+1] = t[n] + dt
        u[n+1] = u[n] + dt*f(t[n], u[n])

    return t, u
```

Eksempel på bruk av funksjonen Forward-Euler

ODE: $u'(t) = u(t), \ u(0) = 1$

Vi går frem som følger:

```
# Vi definerer høyresiden i likningen som en funksjon f(t,u)

def f(t,u):
   return u

# Vi kaller på ForwardEuler-funksjonen


t,u  = ForwardEuler(f, U0=1, T=3, N=30)

# Vi plotter resultatet

import matplotlib.pyplot as plt
plt.plot(t,u)
plt.show()
```

Lær deg å finne $f(u, t)$

Hva er $f(\mathrm{t,u})$ i hvert av disse tilfellene?

Eksempel A:

$$u'(t) = a\,u(t) - b\,u(t)^2$$

Eksempel B:

$$u'(t) - a\,u(t) = b \sin t$$

Eksempel C:

$$\frac{\omega'(t)}{\omega(t)} = a\,(1 - \frac{b}{\omega(t)})$$

Løsning

Hva er $f(\mathrm{t},\mathrm{u})$ i hvert av disse tilfellene?

Eksempel A:

$$u'(t) = a\,u(t) - b\,u(t)^2 \longrightarrow f(\mathrm{t},\mathrm{u}) = a\,u - b\,u^2$$

Eksempel B:

$$u'(t) - a\,u(t) = b \sin t \longrightarrow f(\mathrm{t},\mathrm{u}) = a\,u + b \sin t$$

Eksempel C:

$$\frac{\omega'(t)}{\omega(t)} = a\,(1 - \frac{b}{\omega(t)}) \longrightarrow f(\mathrm{t},\mathrm{u}) = a\,u\,(1 - b/u)$$

Høyereordens ODE'er

En høyereordens ODE er en differensiallikning som også har med $u''(t)$ eller andre deriverte av $u(t)$.

Betrakt likningen $y''(t) = y(t)$. Vi kan omskrive den til et likningssystem med to ukjente:

$$\begin{aligned} x'(t) &= y(t) \\ y'(t) &= x(t) \end{aligned}$$

Dette kan også skrives som

$$\begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix} = \begin{pmatrix} y(t) \\ x(t) \end{pmatrix}$$

eller mer kompakt:

$$u'(t) = f(t, u(t))$$

Eksempel på en **vektor-ODE** og tema for neste forelesning.

Svakheter med ForwardEuler-funksjonen

Selv om ForwardEuler-funksjonen fungerer fint, skal vi se at det er bedre å implementere den som en *klasse*. Vi ønsker at vi skal kunne løse en likning slik:

```
fe = ForwardEuler(f)
fe.set_initial_condition(0.5)
t,u = fe.solve(np.linspace(0, 5, 100))
```

Klasse-implementasjon av Forward Euler (første versjon)

```
import numpy as np

class ForwardEuler:
   def __init__(self, f):
      self.f = f

   def set_initial_condition(self, U0):
      self.U0 = U0

   def solve(self, time_points):
      n = time_points.size
      t = time_points
      u = np.zeros(n)
      u[0] = self.U0  # Sett initialbetingelse
      for k in range(n-1): # Finn u[1], u[2], ...
         dt = t[k+1]-t[k]
      u[k+1] = u[k] + dt * f(t[k], u[k])
      return t, u
```

Klasse-implementasjon av Forward Euler (endelig versjon)

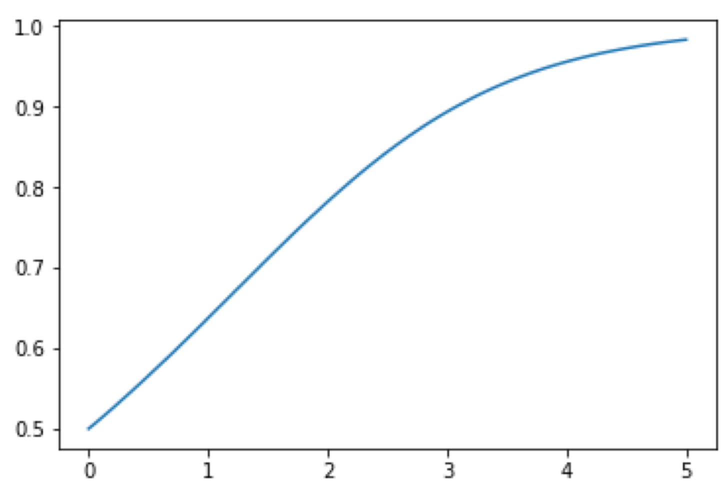
```
import numpy as np

class ForwardEuler:
   def __init__(self, f):
      self.f = f

   def set_initial_condition(self, U0):
      self.U0 = U0

   def solve(self, time_points):
      n = time_points.size
      self.t = time_points
      self.u = np.zeros(n)
      self.u[0] = self.U0
      for k in range(n-1):
         self.k = k
         self.u[k+1] = self.advance()
      return self.u, self.t

   def advance(self):
      u=self.u; t=self.t; f=self.f; k=self.k
      dt = t[k+1]-t[k]
      return u[k] + dt * f(t[k], u[k])
```

Eksempel på bruk

ODE: $u'(t) = u(t)^2(1 - u(t)),\ u(0) = 0.5$

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

# Definer f(t,u):
f = lambda t,u: u**2 * (1-u)

# Opprett instans av likningsløseren:
fe = ForwardEuler(f)

# Sett initialbetingelsen:
fe.set_initial_condition(0.5)

# Velg et grid av t-verdier og løs likningen:
t = np.linspace(0, 5, 100)
t,u = fe.solve(t)

# Plott løsningen
plt.plot(t,u)
```

Eksempel på bruk



Andre løsningsmetoder

Det finnes mange andre løsningsmetoder enn Forward Euler!
Ett eksempel er 4.de ordens Runge-Kutta-metoden:

$u_{k+1} = u_k + \frac{1}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right)$

hvor

$$\begin{aligned}
K_1 &= \Delta t \cdot f(t_k, u_k) \\
K_2 &= \Delta t \cdot f(t_k + \frac{1}{2}\Delta t, u_k + \frac{1}{2}K_1) \\
K_3 &= \Delta t \cdot f(t_k + \frac{1}{2}\Delta t, u_k + \frac{1}{2}K_2) \\
K_4 &= \Delta t \cdot f(t_k + \Delta t, u_k + K_3)
\end{aligned}$$

Hvordan implementere alle metodene i ett program?

Alternativ A

Vi lager én klasse for Forward Euler-metoden (allerede gjort), én klasse for Runge Kutta-metoden, osv.

Svakhet: mye kode vil være helt lik i de to klassene:

- `__init__` blir helt lik
- `set_initial_condition` blir helt lik
- `solve` blir helt lik

Eneste forskjell: metoden `advance`.

Dette peker mot å lagre all felles funksjonalitet i en superklasse `ODESolver` og så bruke to subklasser til å definere de to `advance` metodene.

Alternativ B

```
class ODESolver:
   def __init__(self, f):
      ...osv...

   def set_initial_condition(self, U0):
      ...osv...

   def solve(self, time_points):
      ...osv...

class ForwardEuler(ODESolver):
   def advance(self):
      u=self.u; t=self.t; f=self.f; k=self.k
      dt = t[k+1]-t[k]
      return u[k] + dt * f(t[k], u[k])

class RungeKutta4(ODESolver):
   def advance(self):
      ....osv...
```

ODESolver

Lenke til ODESolver finner du under *Beskjeder* på kursets nettsider.