Objektorientert programmering og løsning av ODE'er

Ole Christian Lingjærde, Institutt for Informatikk, UiO

31. oktober 2019

Dagens tema

- Hva er en differensiallikning?
- Eksempler på anvendelser
- Hvordan kan vi løse dem med datamaskin?
- Pakken ODESolver

Eksempel A

Anta at vi har likningen:

$$u'(t) = 1$$

Her ser vi at:

- For alle t er stigningen til kurven lik 1
- Intuisjon: en rett linje
- Matematisk løsning: $u(t) = t + C$
- Hvis $u(0)$ er kjent kan vi regne ut C

Eksempel A

Anta at vi har likningen:

$$u'(t) = 1$$

Her ser vi at:

- For alle t er stigningen til kurven lik 1
- Intuisjon: en rett linje
- Matematisk løsning: $u(t) = t + C$
- Hvis $u(0)$ er kjent kan vi regne ut C

Eksempel A

Anta at vi har likningen:

$$u'(t) = 1$$

Her ser vi at:

- For alle t er stigningen til kurven lik 1
- Intuisjon: en rett linje
- Matematisk løsning: $u(t) = t + C$
- Hvis $u(0)$ er kjent kan vi regne ut C

Eksempel A

Anta at vi har likningen:

$$u'(t) = 1$$

Her ser vi at:

- For alle t er stigningen til kurven lik 1
- Intuisjon: en rett linje
- Matematisk løsning: $u(t) = t + C$
- Hvis $u(0)$ er kjent kan vi regne ut C

Eksempel A

Anta at vi har likningen:

$$u'(t) = 1$$

Her ser vi at:

- For alle t er stigningen til kurven lik 1
- Intuisjon: en rett linje
- Matematisk løsning: $u(t) = t + C$
- Hvis $u(0)$ er kjent kan vi regne ut C

Eksempel B

Anta at vi har likningen:

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

Her kan vi konkludere:

- For alle t er stigningen til kurven lik høyden
- Intuisjon: en funksjon som vokser raskere og raskere
- Matematisk løsning: $u(t) = A \cdot e^t$.
- Hvis $u(0)$ er kjent kan vi finne A.

Eksempel B

Anta at vi har likningen:

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

Her kan vi konkludere:

- For alle t er stigningen til kurven lik høyden
- Intuisjon: en funksjon som vokser raskere og raskere
- Matematisk løsning: $u(t) = A \cdot e^t$.
- Hvis $u(0)$ er kjent kan vi finne A.

Eksempel B

Anta at vi har likningen:

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

Her kan vi konkludere:

- For alle t er stigningen til kurven lik høyden
- Intuisjon: en funksjon som vokser raskere og raskere
- Matematisk løsning: $u(t) = A \cdot e^t$.
- Hvis $u(0)$ er kjent kan vi finne A.

Eksempel B

Anta at vi har likningen:

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

Her kan vi konkludere:

- For alle t er stigningen til kurven lik høyden
- Intuisjon: en funksjon som vokser raskere og raskere
- Matematisk løsning: $u(t) = A \cdot e^t$.
- Hvis $u(0)$ er kjent kan vi finne A.

Eksempel B

Anta at vi har likningen:

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

Her kan vi konkludere:

- For alle t er stigningen til kurven lik høyden
- Intuisjon: en funksjon som vokser raskere og raskere
- Matematisk løsning: $u(t) = A \cdot e^t$.
- Hvis $u(0)$ er kjent kan vi finne A.

Eksempel C

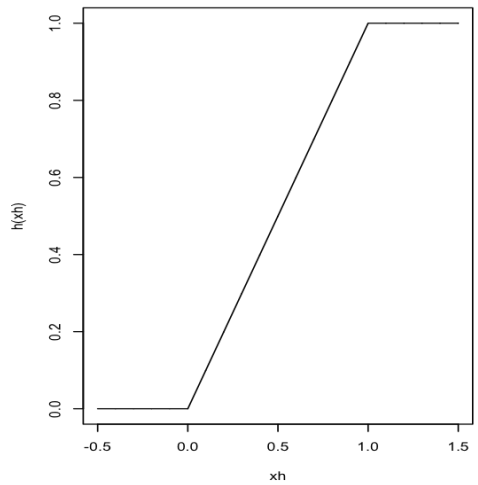
Anta at vi har likningen:

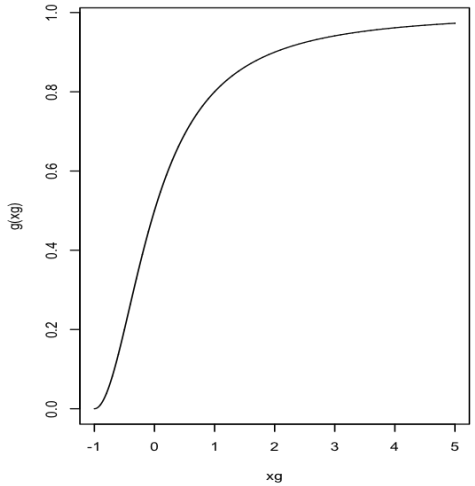
$$u'(t) = u(t)(1 - u(t))$$

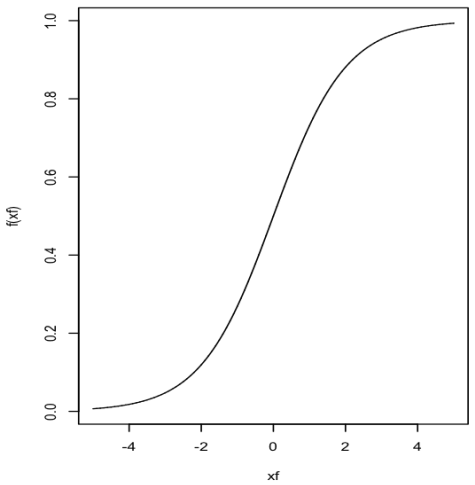
Vi ser at:

- Når $u(t) = 0$ så er $u'(t) = 0$
- Når $u(t) = 1$ så er $u'(t) = 0$
- Når $0 < u(t) < 1$ så er $u'(t) > 0$.

Eksempler på funksjoner som oppfører seg slik:







Numeriske løsninger

I fortsettelsen fokuserer vi på å finne *numeriske* løsninger og vi vil ofte ønske å vise dem *grafisk* også:

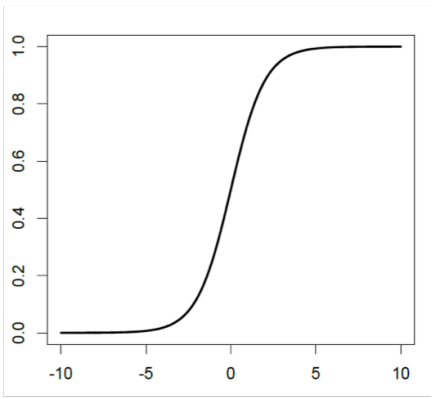
Likning

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

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

Anta at vi har likningen $u'(t) = u(t)$ og initialbetingelsen $u(0) = 1$. Vi ønsker å finne $u(t)$ for $t \in [0, 2]$.

Vi resonnerer som følger:

- La $t_0 = 0$, $t_1 = 0.1$, $t_2 = 0.2$, ..., $t_n = 2.0$.
- Vi vet at
$$u'(t_k) \approx \frac{u(t_{k+1}) - u(t_k)}{t_{k+1} - t_k} = \frac{u(t_{k+1}) - u(t_k)}{0.1}$$
- Dermed har vi tilnærmet (fra likningen) at
$$\frac{u(t_{k+1}) - u(t_k)}{0.1} = u(t_k)$$
- Dette skriver vi lett om til:
$$u(t_{k+1}) = u(t_k) + 0.1 \cdot u(t_k)$$

Løsningsalgoritme

Nå kan vi enkelt finne løsningen!

1) Vi har $u(t_0) = 1$ (initialbetingelsen)
2) Vi får da at

$$u(t_1) = u(t_0) + 0.1 \cdot u(t_0) = 1.0 + 0.1 \cdot 1.0 = 1.1$$

3) Vi får da at

$$u(t_2) = u(t_1) + 0.1 \cdot u(t_1) = 1.1 + 0.1 \cdot 1.1 = 1.21$$

4) ...osv helt opp til $u(t_n)$.

Implementasjon i Python

```python
import numpy as np

# Sett steglengde og max t-verdi
dt = 0.1
T = 2

# Finn antall t-verdier
n = int(T/dt) + 1

# Sett opp arrayer til å holde t- og u-verdier
t = np.linspace(0, T, n)
u = np.zeros(len(t))

# Sett inn initialbetingelsen
u[0] = 1

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

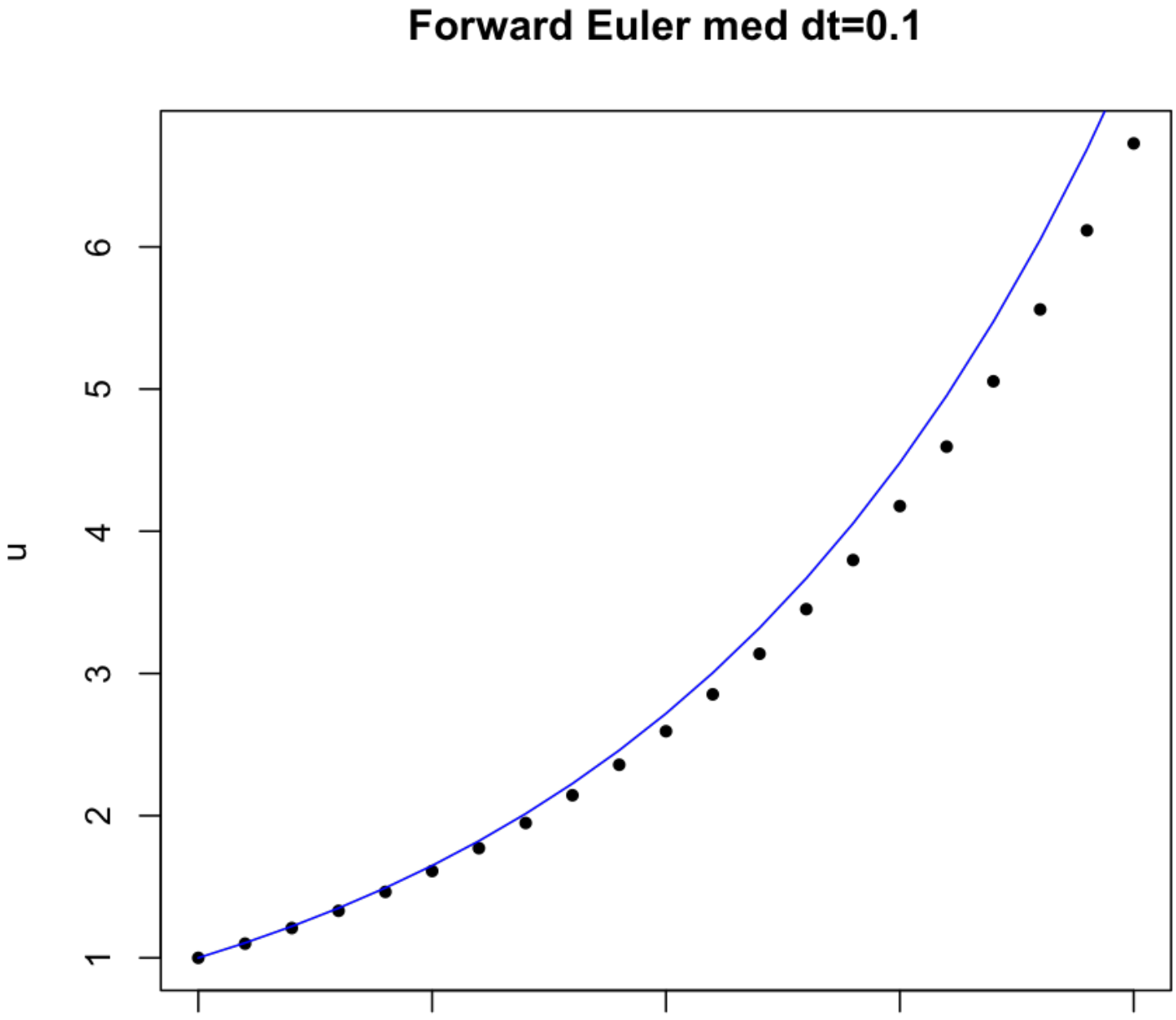
Resultat

```
t                u
0.0          1.000000
0.1          1.100000
0.2          1.210000
0.3          1.331000
0.4          1.464100
0.5          1.610510
0.6          1.771561
....     ....
....     ....
1.8          5.559917
1.9          6.115909
2.0          6.727500
```

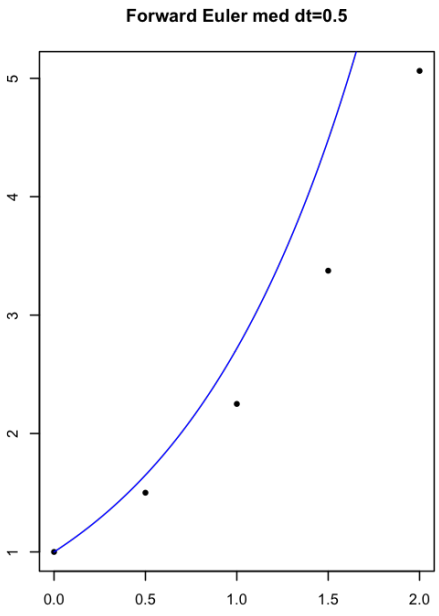
Implementasjon med plotting

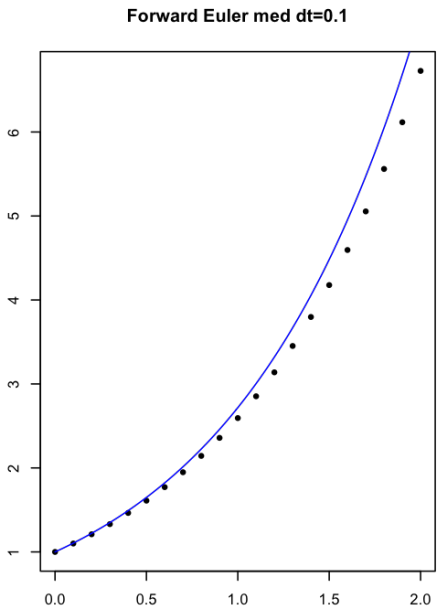
```
import numpy as np
import matplotlib.pyplot as plt

# Sett steglengde og max t-verdi
dt = 0.1
T = 2

# Finn antall t-verdier
n = int(T/dt) + 1

# Sett opp arrayer
t = np.linspace(0, T, n)
u = np.zeros(n)

# Sett inn initialbetingelsen
u[0] = 1

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

# Plott løsningen
plt.plot(t, u, 'ko')
plt.plot(t, np.exp(t), 'b-')
plt.show()
```

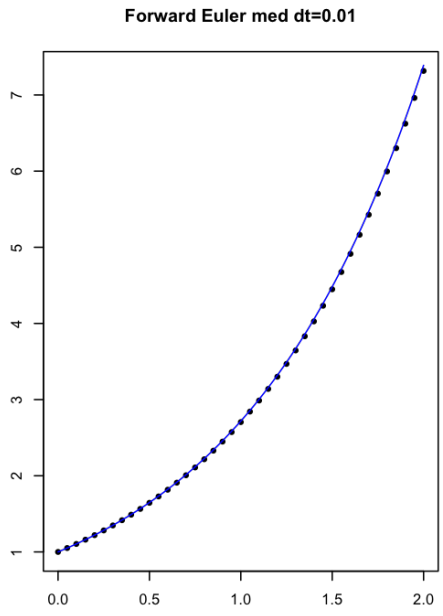
Resultat



Avstanden mellom t_k-verdiene betyr mye!







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

Anta at vi har likningen $u'(t) = u(t)(1 - u(t))$ og initialbetingelsen $u(0) = 0.5$. Vi ønsker å finne $u(t)$ for $t \in [0, 5]$.

Vi resonnerer som i forrige eksempel:

- La $t_0 = 0.0$, $t_1 = 0.1$, $t_2 = 0.2$, ..., $t_n = 5.0$.
- Vi vet at
$$u'(t_k) \approx \frac{u(t_{k+1}) - u(t_k)}{t_{k+1} - t_k} = \frac{u(t_{k+1}) - u(t_k)}{0.1}$$
- Dermed har vi tilnærmet (fra likningen) at
$$\frac{u(t_{k+1}) - u(t_k)}{0.1} = u(t_k)(1 - u(t_k))$$
- Dette skriver vi lett om til:
$$u(t_{k+1}) = u(t_k) + 0.1 \cdot u(t_k)(1 - u(t_k))$$

Løsningsalgoritme

1) Vi har $u(t_0) = 0.5$ (initialbetingelsen)
2) Vi får da at

$$u(t_1) = u(t_0) + 0.1 \cdot u(t_0)(1 - u(t_0)) = 0.5 + 0.1 \cdot 0.5^2 = 0.525$$

3) Vi får da at

$$u(t_2) = u(t_1) + 0.1 \cdot u(t_1)(1 - u(t_1)) = ...$$

4) ...osv helt opp til $u(t_n)$.

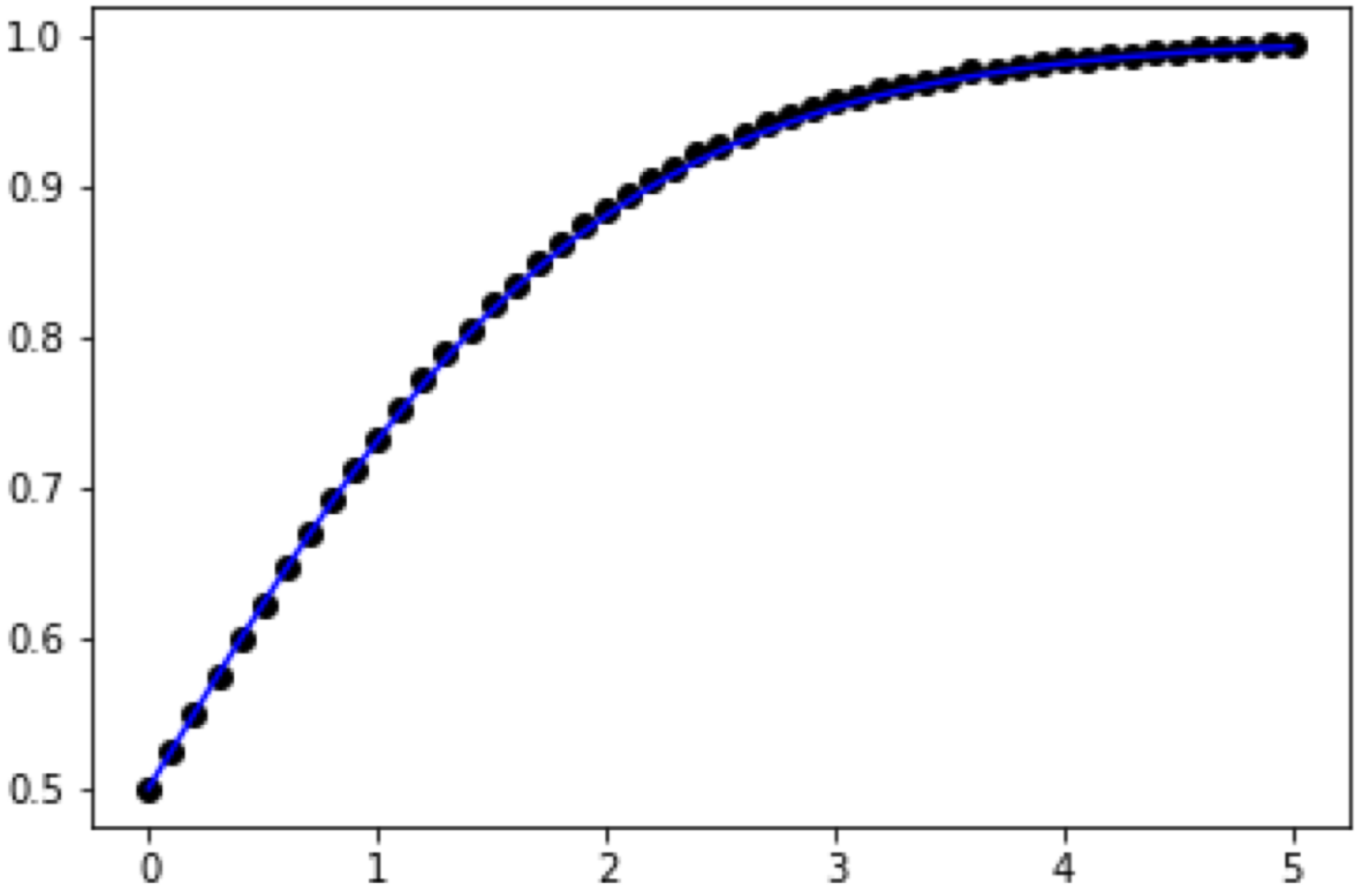
Implementasjon i Python

```python
import numpy as np

# Sett steglengde og max t-verdi
dt = 0.1
T = 5

# Finn antall t-verdier
n = int(T/dt) + 1

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

# Sett inn initialbetingelsen
u[0] = 0.5

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

Resultat

t	u
0.0	0.5000
0.1	0.5250
0.2	0.5499
0.3	0.5747
0.4	0.5991
0.5	0.6231
0.6	0.6466
0.7	0.6695
0.8	0.6916
0.9	0.7129
1.0	0.7334
1.1	0.7530
1.2	0.7716
....	....
....	....
4.5	0.9907
4.6	0.9916
4.7	0.9924
4.8	0.9932
4.9	0.9939
5.0	0.9945

Resultat



Sammenlikning av de to eksemplene

```python
import numpy as np

# Sett steglengde og max t-verdi
dt = 0.1
T = 5

# Finn antall t-verdier
n = int(T/dt) + 1

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

# Sett inn initialbetingelsen
u[0] = 0.5

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

Merk: 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 i in range(1, n):
    u[i] = u[i-1] + dt * u[i]
```

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

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

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

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

Lær deg å se hva $f(u,t)$ er!

Hva er $f(u,t)$ 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:

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

Løsning

Hva er $f(u,t)$ i hvert av disse tilfellene?

Eksempel A:

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

Eksempel B:

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

Eksempel C:

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

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:

$$\mathbf{u}'(t) = \mathbf{f}(\mathbf{u}(t), t)$$

Eksempel på en **vektor-ODE** og er tema for de neste forelesninger.

Forward Euler-metoden

Anta at vi har en ODE $u'(t) = f(u(t), t)$ og en initialbetingelse $u(0) = U_0$. Vi ønsker å finne $u(t)$ for $t \in [0, T]$.

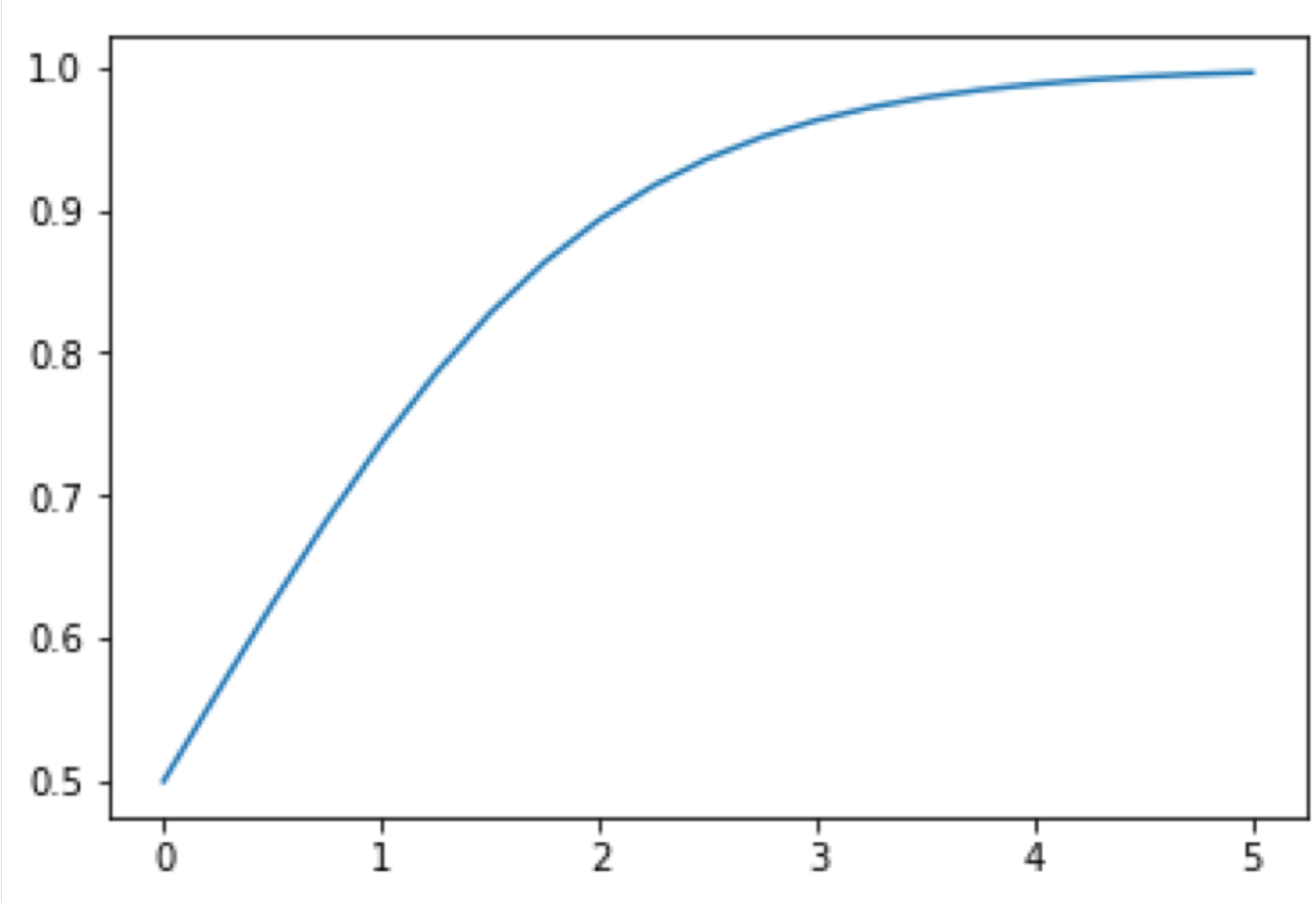
Forward Euler

```
import numpy as np

def ForwardEuler(f, U0, T, n):
    t = np.zeros(n+1)
    u = np.zeros(n+1)
    u[0] = U0
    dt = T/n
    for i in range(n):
        t[i+1] = t[i] + dt
        u[i+1] = u[i] + dt * f(u[i], t[i])
    return u, t
```

Eksempel på bruk

```
import matplotlib.pyplot as plt

def f(u,t):
    return u * (1-u)

u, t = ForwardEuler(f, U0=1, T=4, n=20)
plt.plot(t, u)
```

Resultat



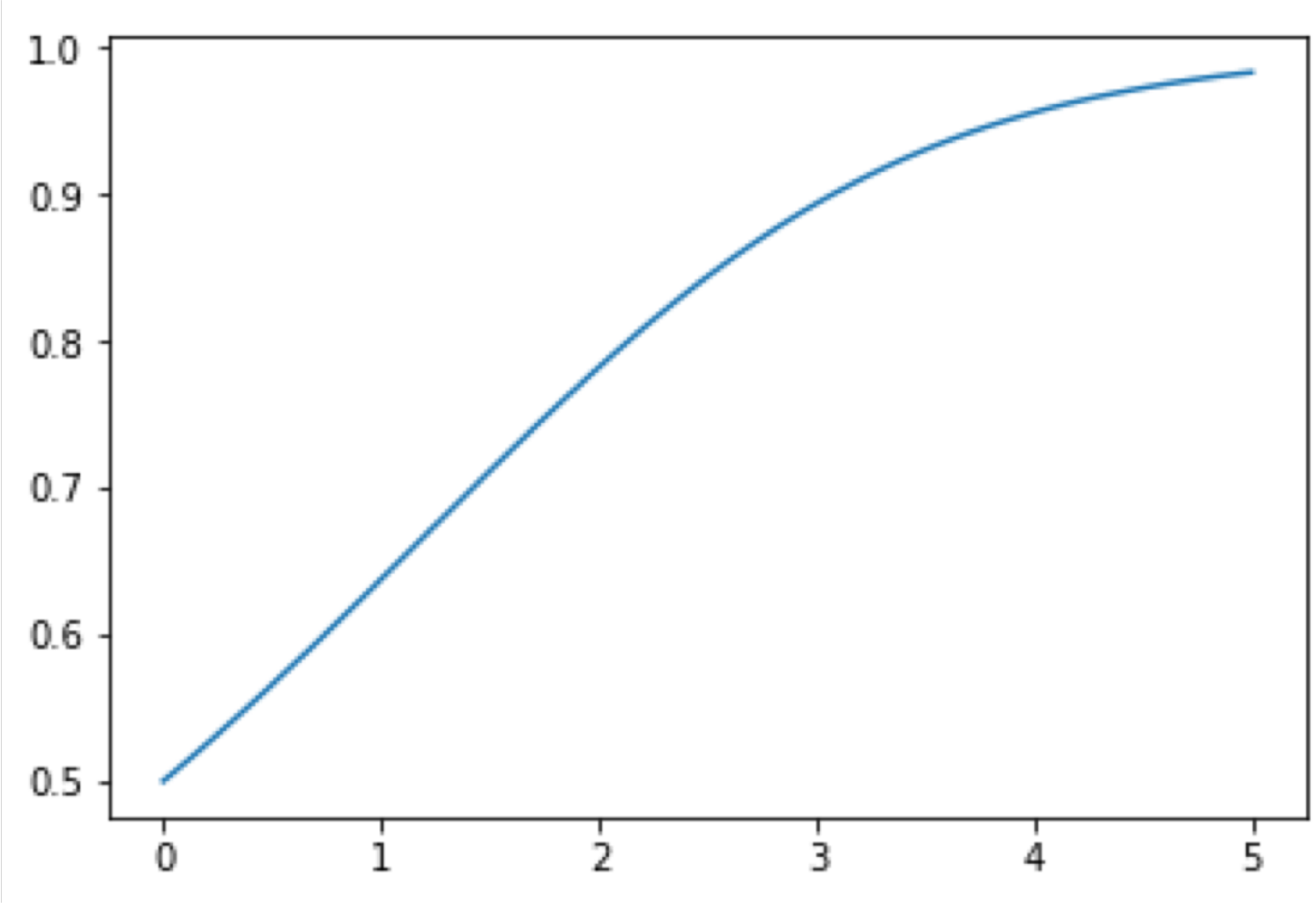
Klasse-implementasjon av Forward Euler

```
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  # Sett init.betingelse
        for k in range(n-1): # Finn u[1], u[2], ...
            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(u[k], t[k])
```

Eksempel på bruk

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

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

# Sett initialbetingelsen:
F.set_initial_condition(0.5)

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

# Plott løsningen
plt.plot(t,u)
```

Eksempel på bruk



Andre løsningsmetoder

Forward Euler-metoden:

$$u_{k+1} = u_k + \Delta t\, f(u_k, t_k)$$

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 $K_1 = \Delta t\, f(u_k, t_k)$, $K_2 = \Delta t\, f(u_k + \frac{1}{2}K_1, t_k + \frac{1}{2}\Delta t)$,
$K_3 = \Delta t\, f(u_k + \frac{1}{2}K_2, t_k + \frac{1}{2}\Delta t)$ og $K_4 = \Delta t\, f(u_k + K3, t_k + \Delta t)$.

Det finnes også en rekke andre metoder. Hvordan kan vi implementere alle metodene i ett program?

Alternativ A

Vi bruker samme fremgangsmåte som hittil, og lager en klasse for Forward Euler-metoden (allerede gjort), en 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(u[k], t[k])

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

ODESolver

Lenke til ODESolver legges ut på nettsiden til kurset!