Forelesning IN1900 – 7 November 2023

Ole Christian Lingjærde
Institutt for informatikk, Universitetet i Oslo

Uke: 6 November – 12 November, 2023

Denne ukens agenda

Skalare ODE'er

Eksempler på løsing av skalare ODE'er

Vektor-ODE'er

Eksempler på løsing av vektor-ODE'er

SIR-modellen

Strukturen til ODESolver

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

  def set_initial_condition(self, u0):
    ...osv...

  def solve(self, t_span, N):
    ...osv...

class ForwardEuler(ODESolver):
  def advance(self):
     u,f,n,t = self.u, self.f, self.n, self.t
     dt = self.dt
     unew = u[n] + dt * f(t[n],u[n])
     return unew

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

Hvordan løse skalare ODE'er med ODESolver

Trinn 1: Finn f(t,u) og implementer denne

Hvordan løse skalare ODE'er med ODESolver

Trinn 1: Finn f(t,u) og implementer denne

$u'(t) = 2u(t)^3$ ➡

```
f = lambda t,u: 2*u**3
```

Hvordan løse skalare ODE'er med ODESolver

Trinn 1: Finn f(t,u) og implementer denne

$u'(t) = 2u(t)^3$ ➡ `f = lambda t,u: 2*u**3`

Trinn 2: Velg løsningsmetode

Hvordan løse skalare ODE'er med ODESolver

Trinn 1: Finn f(t,u) og implementer denne

$u'(t) = 2u(t)^3$ ➡ `f = lambda t,u: 2*u**3`

Trinn 2: Velg løsningsmetode

```
metode = ForwardEuler(f)
```

Hvordan løse skalare ODE'er med ODESolver

Trinn 1: Finn f(t,u) og implementer denne

$u'(t) = 2u(t)^3$ ➡ `f = lambda t,u: 2*u**3`

Trinn 2: Velg løsningsmetode

```
metode = ForwardEuler(f)
```

Trinn 3: Sett initialbetingelse

Hvordan løse skalare ODE'er med ODESolver

Trinn 1: Finn f(t,u) og implementer denne

$u'(t) = 2u(t)^3$ ➡

```
f = lambda t,u: 2*u**3
```

Trinn 2: Velg løsningsmetode

```
metode = ForwardEuler(f)
```

Trinn 3: Sett initialbetingelse

```
metode.set_initial_condition(5.0)
```

Hvordan løse skalare ODE'er med ODESolver

Trinn 1: Finn f(t,u) og implementer denne

$u'(t) = 2u(t)^3$ ➡ `f = lambda t,u: 2*u**3`

Trinn 2: Velg løsningsmetode

```
metode = ForwardEuler(f)
```

Trinn 3: Sett initialbetingelse

```
metode.set_initial_condition(5.0)
```

Trinn 4: Løs likningen

Hvordan løse skalare ODE'er med ODESolver

Trinn 1: Finn f(t,u) og implementer denne

$u'(t) = 2u(t)^3$ ➡ `f = lambda t,u: 2*u**3`

Trinn 2: Velg løsningsmetode

```
metode = ForwardEuler(f)
```

Trinn 3: Sett initialbetingelse

```
metode.set_initial_condition(5.0)
```

Trinn 4: Løs likningen

```
t,u = metode.solve((0,5), 500)
```

Eksempel 1

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

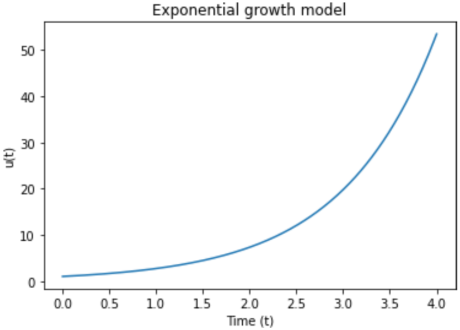
```
from ODESolver import *

# Vi har f(t,u)=u
f = lambda t,u: u

# Vi bruker Forward Euler
metode = ForwardEuler(f)

# Vi setter initialbetingelsen
metode.set_initial_condition(1)

# Vi løser likningen
t,u = metode.solve((0,4), 400)
```

Vi plotter løsningen

```
import matplotlib.pyplot as plt

plt.plot(t,u)
plt.title("Exponential growth model")
plt.xlabel("Time(t)")
plt.ylabel("u(t)")
plt.show()
```

Vi plotter løsningen

```
import matplotlib.pyplot as plt

plt.plot(t,u)
plt.title("Exponential growth model")
plt.xlabel("Time(t)")
plt.ylabel("u(t)")
plt.show()
```



Eksempel 2

ODE: $u'(t) = \alpha\sqrt{u(t)} * (1 - u(t)/R)$, $u(0) = 0.01$

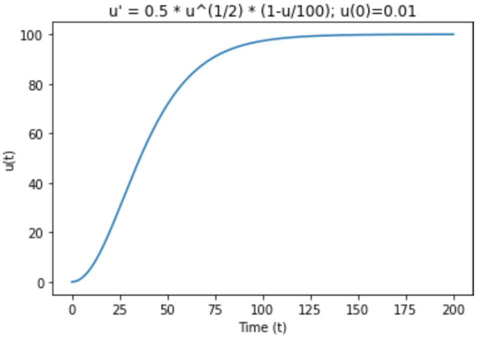
```
from ODESolver import *
import numpy as np
import matplotlib.pyplot as plt

class Fnc:
  def __init__(self, alpha, R):
    self.alpha = alpha
    self.R = R

  def __call__(self, t, u):
    return self.alpha * np.sqrt(u) * (1-u/self.R)

metode = ForwardEuler(Fnc(alpha=0.5, R=100))
metode.set_initial_condition(0.01)
t,u = metode.solve((0,200), 400)
```

Eksempel 2

```
plt.plot(t,u)
plt.title("u' = 0.5 * u^(1/2) * (1-u/100); u(0)=0.01")
plt.xlabel("Time(t)")
plt.ylabel("u(t)")
plt.show()
```

Eksempel 2

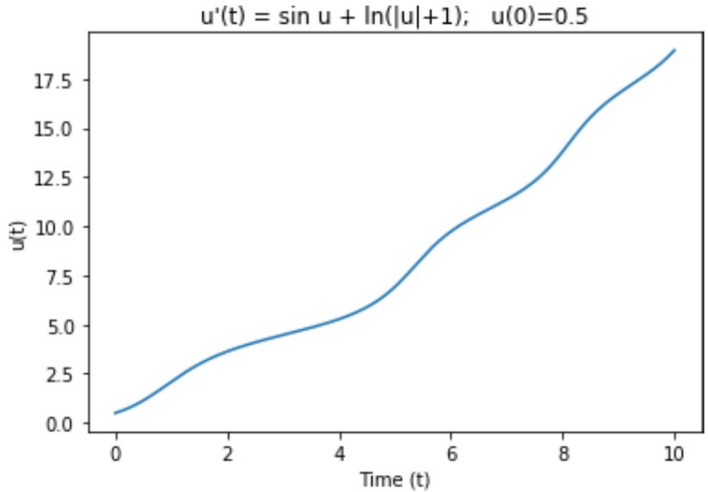
```
plt.plot(t,u)
plt.title("u' = 0.5 * u^(1/2) * (1-u/100); u(0)=0.01")
plt.xlabel("Time(t)")
plt.ylabel("u(t)")
plt.show()
```



Eksempel 3

ODE: $u'(t) = \sin u(t) + \ln(|u(t)| + 1),\ u(0) = 0.5$

```
from ODESolver import *
import numpy as np

# Implementerer f(t,u)
f = lambda t,u: np.sin(u) + np.log(abs(u)+1)

# Velger likningsløseren RungeKutta4
metode = RungeKutta4(f)

# Setter initialbetingelsen
metode.set_initial_condition(0.5)

# Løser likningen
t,u = metode.solve((0,10), 500)
```


u'(t) = sin u + ln(|u|+1);   u(0)=0.5
u(t)
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
2
4
6
8
10
Time (t)

Anta at vi har følgende system hvor $x(t)$ og $y(t)$ er ukjente:

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

Vi vil finne løsning for $t \in [0, 6]$ når $x(0) = 1$, $y(0) = 0$:

Anta at vi har følgende system hvor $x(t)$ og $y(t)$ er ukjente:

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

Vi vil finne løsning for $t \in [0, 6]$ når $x(0) = 1$, $y(0) = 0$:

- La $t_k = k \cdot dt$, $k = 0, 1, \ldots, n$.

Anta at vi har følgende system hvor $x(t)$ og $y(t)$ er ukjente:

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

Vi vil finne løsning for $t \in [0, 6]$ når $x(0) = 1$, $y(0) = 0$:

- La $t_k = k \cdot dt$, $k = 0, 1, \ldots, n$.
- Hvis $y(t)$ antas kjent kan vi finne $x(t)$ med Forward-Euler:

$$x(t_{k+1}) = x(t_k) + dt \cdot y(t_k)$$

Anta at vi har følgende system hvor $x(t)$ og $y(t)$ er ukjente:

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

Vi vil finne løsning for $t \in [0, 6]$ når $x(0) = 1$, $y(0) = 0$:

- La $t_k = k \cdot dt$, $k = 0, 1, \ldots, n$.
- Hvis $y(t)$ antas kjent kan vi finne $x(t)$ med Forward-Euler:

$$x(t_{k+1}) = x(t_k) + dt \cdot y(t_k)$$

- Hvis $x(t)$ antas kjent kan vi finne $y(t)$ med Forward-Euler:

$$y(t_{k+1}) = y(t_k) - dt \cdot x(t_k)$$

Anta at vi har følgende system hvor $x(t)$ og $y(t)$ er ukjente:

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

Vi vil finne løsning for $t \in [0, 6]$ når $x(0) = 1$, $y(0) = 0$:

- La $t_k = k \cdot dt$, $k = 0, 1, \ldots, n$.
- Hvis $y(t)$ antas kjent kan vi finne $x(t)$ med Forward-Euler:

$$x(t_{k+1}) = x(t_k) + dt \cdot y(t_k)$$

- Hvis $x(t)$ antas kjent kan vi finne $y(t)$ med Forward-Euler:

$$y(t_{k+1}) = y(t_k) - dt \cdot x(t_k)$$

- På vektorform:

$$\begin{pmatrix} x(t_{k+1}) \\ y(t_{k+1}) \end{pmatrix} = \begin{pmatrix} x(t_k) \\ y(t_k) \end{pmatrix} + dt \cdot \begin{pmatrix} y(t_k) \\ -x(t_k) \end{pmatrix}$$

Oppdateringsregelen:

$$\begin{pmatrix} x(t_{k+1}) \\ y(t_{k+1}) \end{pmatrix} = \begin{pmatrix} x(t_k) \\ y(t_k) \end{pmatrix} + dt \cdot \begin{pmatrix} y(t_k) \\ -x(t_k) \end{pmatrix}$$

Oppdateringsregelen:

$$\begin{pmatrix} x(t_{k+1}) \\ y(t_{k+1}) \end{pmatrix} = \begin{pmatrix} x(t_k) \\ y(t_k) \end{pmatrix} + dt \cdot \begin{pmatrix} y(t_k) \\ -x(t_k) \end{pmatrix}$$

Anta at vi definerer:

$$\mathbf{u}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}, \quad \mathbf{f}(t, \mathbf{u}) = \begin{pmatrix} u_2 \\ -u_1 \end{pmatrix}$$

Oppdateringsregelen:

$$\begin{pmatrix} x(t_{k+1}) \\ y(t_{k+1}) \end{pmatrix} = \begin{pmatrix} x(t_k) \\ y(t_k) \end{pmatrix} + dt \cdot \begin{pmatrix} y(t_k) \\ -x(t_k) \end{pmatrix}$$

Anta at vi definerer:

$$\mathbf{u}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}, \ \ \mathbf{f}(t, \mathbf{u}) = \begin{pmatrix} u_2 \\ -u_1 \end{pmatrix}$$

Da blir:

$$\mathbf{u}(t_{k+1}) = \mathbf{u}(t_k) + dt \cdot \mathbf{f}(t_k, \mathbf{u}(t_k))$$

ODE-system:

$$\begin{aligned} x'(t) &= y(t) \quad , \quad x(0) = 0 \\ y'(t) &= -x(t) \quad , \quad y(0) = 1 \end{aligned}$$

ODE-system:

$$\begin{aligned} x'(t) &= y(t) \quad , \quad x(0) = 0 \\ y'(t) &= -x(t) \quad , \quad y(0) = 1 \end{aligned}$$

Definerer først **u**(t) slik:

u(t) = (x(t), y(t))

ODE-system:

$$\begin{aligned} x'(t) &= y(t) \quad , \quad x(0) = 0 \\ y'(t) &= -x(t) \quad , \quad y(0) = 1 \end{aligned}$$

Definerer først **u**(t) slik:

$\mathbf{u}(t) = (x(t), y(t))$

Da blir

$\mathbf{u}'(t) \quad = \quad (x'(t), y'(t))$

ODE-system:

$$\begin{aligned} x'(t) &= y(t) \quad , \quad x(0) = 0 \\ y'(t) &= -x(t) \quad , \quad y(0) = 1 \end{aligned}$$

Definerer først **u**(t) slik:

u(t) = (x(t), y(t))

Da blir

u'(t) = (x'(t), y'(t)) = (y(t), -x(t))

ODE-system:

$$\begin{aligned} x'(t) &= y(t) \quad , \quad x(0) = 0 \\ y'(t) &= -x(t) \quad , \quad y(0) = 1 \end{aligned}$$

Definerer først **u**(t) slik:

$\mathbf{u}(t) = (x(t), y(t))$

Da blir

$$\mathbf{u}'(t) = (x'(t), y'(t)) = (y(t), -x(t)) = (u_2(t), -u_1(t))$$

ODE-system:

$$\begin{aligned} x'(t) &= y(t) \quad , \quad x(0) = 0 \\ y'(t) &= -x(t) \quad , \quad y(0) = 1 \end{aligned}$$

Definerer først **u**(t) slik:

$\mathbf{u}(t) = (x(t), y(t))$

Da blir

$\mathbf{u}'(t) \quad = \quad (x'(t), y'(t)) \quad = \quad (y(t), -x(t)) \quad = \quad (u_2(t), -u_1(t))$

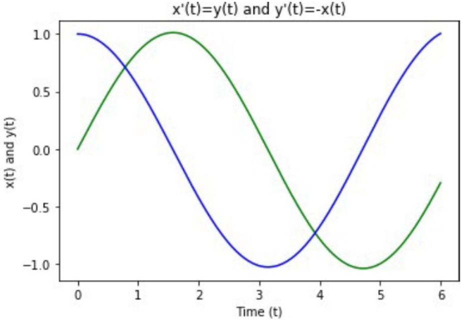
Dvs

$\mathbf{u}'(t) \quad = \quad f(t, \mathbf{u}(t)) \qquad \text{hvor} \qquad \mathbf{f}(t,\mathbf{u}) = (u_2, -u_1)$

```
from ODESolver import *
import numpy as np

# Implementerer f(t,u)
f = lambda t,u: np.array([u[1],-u[0]])

# Velger likningsløseren Forward Euler
metode = ForwardEuler(f)

# Setter initialbetingelsen
metode.set_initial_condition([0,1])

# Løser likningen
t,u = metode.solve((0,6), 400)
```

```python
import matplotlib.pyplot as plt

plt.plot(t,u)
plt.title("x'(t)=y(t) and y'(t)=-x(t)")
plt.xlabel("Time(t)")
plt.ylabel("x(t) and y(t)")
plt.show()
```

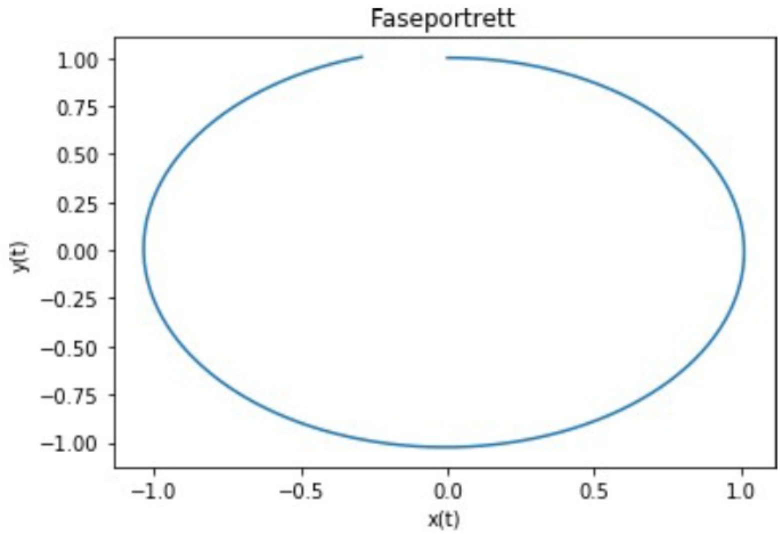
```
import matplotlib.pyplot as plt

plt.plot(t,u)
plt.title("x'(t)=y(t) and y'(t)=-x(t)")
plt.xlabel("Time(t)")
plt.ylabel("x(t) and y(t)")
plt.show()
```



Faseportretter

I stedet for å plotte x(t) og y(t) som funksjon av tiden, kan vi plotte x(t) og y(t) mot hverandre. Det kalles et *faseportrett*.

```
import matplotlib.pyplot as plt
plt.plot(u[:,0], u[:,1])
plt.title("Faseportrett")
plt.xlabel("x(t)")
plt.ylabel("y(t)")
plt.show()
```

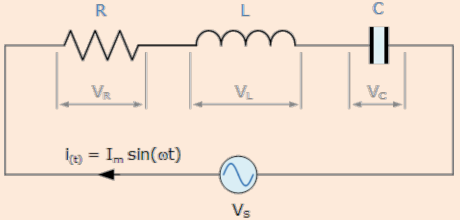


Faseportrett
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y(t)
−1.0
−0.5
0.0
0.5
1.0
x(t)

ODE-system (*van der Pol likningen*):

$$
\begin{aligned}
x'(t) &= y(t) - x(t)^3 + x(t) \quad , \quad x(0) = 0.1 \\
y'(t) &= -x(t) \quad , \quad y(0) = 0
\end{aligned}
$$

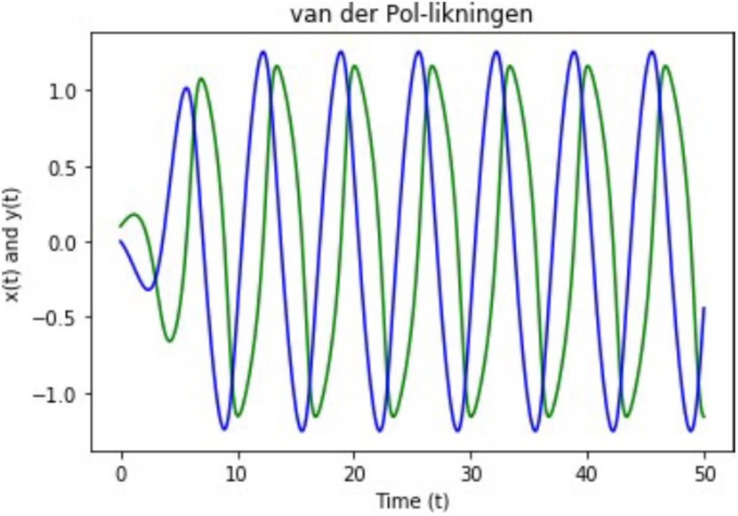
Likninger som beskriver strøm og spenning i en elektrisk RLC-krets under bestemte betingelser.

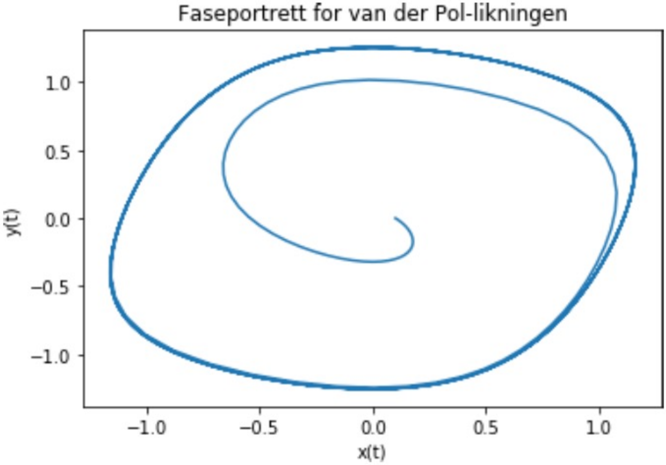


ODE-system (*van der Pol likningen*):

$$
\begin{aligned}
x'(t) &= y(t) - x(t)^3 + x(t) \quad , \quad x(0) = 0.1 \\
y'(t) &= -x(t) \quad , \quad y(0) = 0
\end{aligned}
$$

```
from ODESolver import *
import numpy as np

def f(t,u):
   x,y = u[0],u[1]
   return [y - x**3 + x, -x]

metode = RungeKutta4(f)
metode.set_initial_condition([0.1,0.0])
t,u = metode.solve((0,50), 400)
```


van der Pol-likningen
1.0
0.5
0.0
−0.5
−1.0
x(t) and y(t)
0
10
20
30
40
50
Time (t)


Faseportrett for van der Pol-likningen
1.0
0.5
0.0
−0.5
−1.0
y(t)
−1.0
−0.5
0.0
0.5
1.0
x(t)

ODE-system (*Lorenz-systemet*):

$$
\begin{aligned}
x' &= 10(y - x) \\
y' &= 28x - y - xz \\
z' &= xy - (8/3)z
\end{aligned}
$$

Forsøk på å forklare noe av den uforutsigbare oppførselen til været.
Tenk deg en planet hvor atmosfæren består av én væskepartikkel:

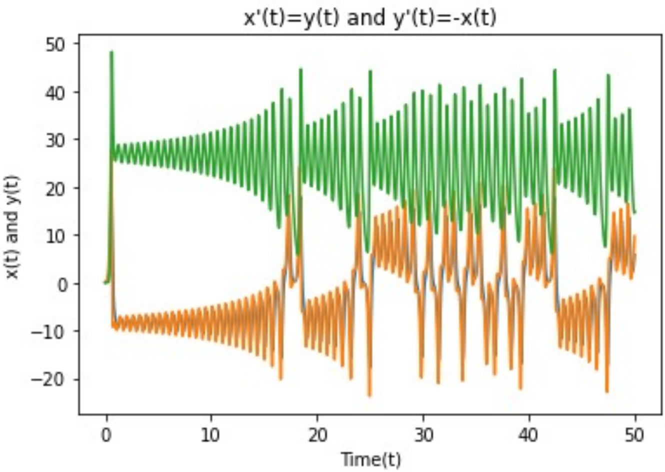
- partikkelen varmes opp fra bakken og stiger
- partikkelen kjøles ovenfra og synker

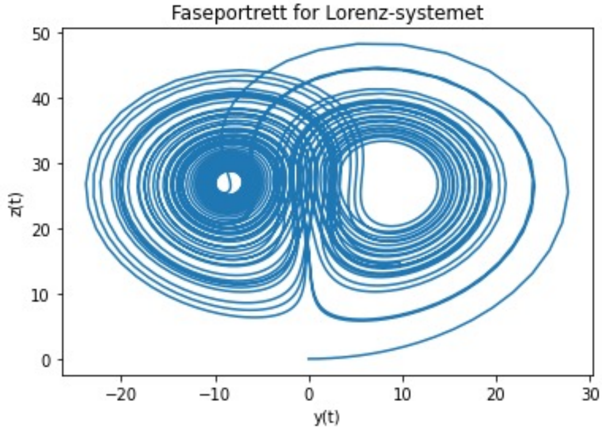
Kan vi predikere "været" på denne planeten?

ODE-system (*Lorenz-systemet*):

$$\begin{aligned} x' &= 10(y-x) \\ y' &= 28x - y - xz \\ z' &= xy - (8/3)z \end{aligned}$$

```
from ODESolver import *
import numpy as np

def f(t,u):
  x,y,z = u[0],u[1],u[2]
  return np.array([10*(y-x), 28*x-y-x*z, x*y-(8/3)*z])

metode = RungeKutta4(f)
metode.set_initial_condition([0.1,0.0,0.0])
t,u = metode.solve((0,50), 4000)
```

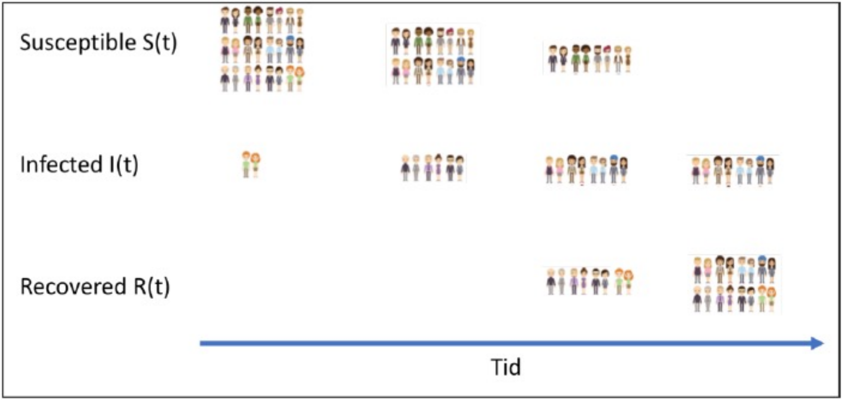

x'(t)=y(t) and y'(t)=-x(t)
x(t) and y(t)
Time(t)
50
40
30
20
10
0
−10
−20
0
10
20
30
40
50



Faseportrett for Lorenz-systemet
50
40
30
20
10
0
z(t)
−20
−10
0
10
20
30
y(t)

Forestill deg at du skal lage en datasimulering over hvordan et utbrudd av en smittsom sykdom utvikler seg over tid. Det er nyttig å holde rede på tre tall på hvert tidspunkt:

- $S(t)$: antall i faresone for å bli smittet
- $I(t)$: antall smittede og smittefarlige
- $R(t)$: antall som har blitt friske igjen


Susceptible S(t)
Infected I(t)
Recovered R(t)
Tid

Dette er en modell for hvordan $S(t)$, $I(t)$ og $R(t)$ endrer seg over tid under et sykdomsutbrudd. Modellen består av tre ODE'er:

$$
\begin{aligned}
S'(t) &= -\beta S(t)I(t) \\
I'(t) &= \beta S(t)I(t) - \nu I(t) \\
R'(t) &= \nu I(t)
\end{aligned}
$$

Disse likningene sier:

- Antall som smittes ved tid t er $\beta S(t)I(t)$
- Dette kommer til fratrekk på $S(t)$ og tilskudd på $I(t)$
- En viss prosent av de smittede blir friske igjen
- Dette kommer til fratrekk på $I(t)$ og tilskudd på $R(t)$

Siden SIR-modellen har to parametre (β og ν) implementerer vi den som en klasse:

```
class SIR:
  def __init__(self, beta, nu):
     self.beta = beta
     self.nu = nu

  def __call__(self, t, u):
     beta,nu = self.beta,self.nu
     S,I,R = u[0],u[1],u[2]
     f1 = -beta * S * I
     f2 = beta * S * I - nu * I
     f3 = nu * I
     return [f1,f2,f3]
```

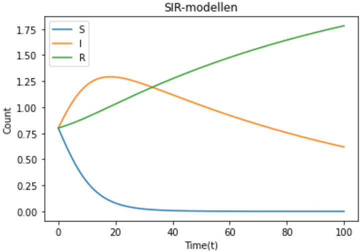
Nå kan vi lett lage modeller, f.eks. f = SIR(0.1, 0.01)

Vi løser SIR-modellen:

```
from ODESolver import *

class SIR:
   ... se forrige slide ...

f = SIR(beta=0.1, nu=0.01)
metode = RungeKutta4(f)
metode.set_initial_condition([0.8, 0.8, 0.8])
t,u = metode.solve((0,100), 500)
```

Vi plotter SIR-modellen:

```
import matplotlib.pyplot as plt

plt.plot(t,u)
plt.title("SIR-modellen")
plt.xlabel("Time(t)")
plt.ylabel("Count")
plt.legend(['S','I','R'])
plt.show()
```



En utvidelse

Valget av β og ν i SIR-modellen bestemmer hvordan funksjonene S(t), I(t) og R(t) blir:

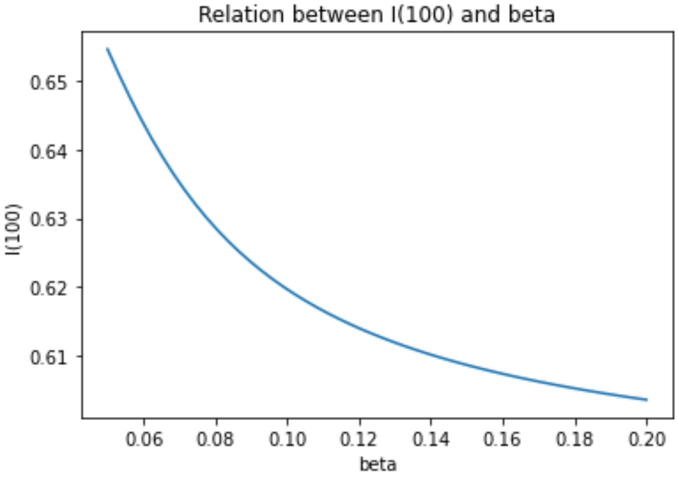
β , ν $\longrightarrow$ S(t), I(t), R(t)

Endrer vi på β så vil vi se en endring i f.eks.

I(100) = antall infiserte etter 100 dager

Men <u>hvordan</u> endrer I(100) seg?

```
from ODESolver import *
import numpy as np

class SIR:
  ... se forrige slide ...

betas = np.linspace(0.05, 0.20, 50)
I_100 = np.zeros_like(betas)
for i in range(len(betas)):
   f = SIR(beta=betas[i], nu=0.01)
   metode = RungeKutta4(f)
   metode.set_initial_condition([0.8,0.8,0.8])
   t,u = metode.solve((0,100), 500)
   I_100[i] = u[499,1]
```


Relation between I(100) and beta
I(100)
0.65
0.64
0.63
0.62
0.61
beta
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20