# Prosjekt 1, Double Pendulum

# Problemstilling

- Målet med dette prosjektet er å designe en klasse i python der vi kan simulere bevegelsen til en dobbel pendel.
- Oppgaven er delt i 4 deler med deloppgaver under hver del.
- Part 1: Solving initial value problem
- Part 2: The single pendulum
- Part 3: The double pendulum
- Part 4: Animating the pendulum

# Metode og utfordringer

## Part 1 Exp decay

```
import math as m
import scipy
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

class ExponentialDecay:
    def __init__(self, a,h=0.1):
        self.a = a
        if self.a <0:
            raise ValueError

    def __call__(self,t,u):
        self.t = t
        self.u = u

        #self.h = 0.001
        return -self.a*u

    def solve(self,u0,T):
        self.u0 = u0
        self.T = T
        sol =  solve_ivp( self,y0=self.u0,t_span = [0,self.T])
        t = sol.t
        u = sol.y
        return t,u


if __name__ == "__main__":
    a = 0.5
    exp_decay = ExponentialDecay(a)
    T = 10
    t,u = exp_decay.solve([2],T)
    plt.plot(t, u[0])
    plt.show()
```

## Part 2 Single pendulum

```
import numpy as np
import scipy
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

#Defining Error
class ODEsNotSolved(Exception):
    pass

#Defining pendulum class
class Pendulum:

    #constructor
    def __init__(self,L,M,g):

        self.L = L #meter
        self.M = M #kg
        self.g = g #m/s^2
        self._theta = None
        self._omega = None
        self._t = None

    #RHS
    def __call__(self, t, y):
        theta, omega = y
        dthetaDT = omega
        domegaDT = (-self.g/self.L)*np.sin(theta)
        return dthetaDT,domegaDT


    def solve(self,y0,T,dt, angles = None):
        self.y0 = y0
        self.T = T
        self.dt = dt
        if angles == "Deg":
            self.y0 = np.radians(self.y0)

        sol =  solve_ivp( self,y0=self.y0,t_span = [0,self.T], max_step= self.dt)

        #Storing private variables instead of returning
        self._theta = sol.y[0]
        self._omega = sol.y[1]
        self._t = sol.t
```

```
#Getting private variables using property decorator
    @property
    def theta(self):
        if self._theta is None:
            raise ODEsNotSolved("No solution found, please remember to call solve")
        return self._theta

    @property
    def omega(self):
        if self._theta is None:
            raise ODEsNotSolved("No solution found, please remember to call solve")
        return self._omega

    @property
    def t(self):
        if self._theta is None:
            raise ODEsNotSolved("No solution found, please remember to call solve")
        return self._t
    @property
    def x(self):
        return self.L*np.sin(self._theta)
    @property
    def y(self):
        return -self.L*np.cos(self._theta)

    @property
    def potential(self):
        pot = self.M*self.g*(self.y+L)
        return pot


    @property
    def vx(self):
        return np.gradient(self.x,self.dt)

    @property
    def vy(self):
        return np.gradient(self.y, self.dt)
    @property
    def kinetic(self):
        kin = 0.5*self.M*(self.vx**2 +self.vy**2)
        return kin
    @property
    def totalE(self):
        return self.kinetic+self.potential
```

# Metode og utfordringer

## Part 3 Double pendulum

```
#Defining DoublePendulum claas
class DoublePendulum():

    #constructor for lengths, gravitational accelration and mass.
    #Optimally we would set M_1 and M_2, but in our case M_1=M_2, also this
    #is the case for L_1 = L_2, but we have defined them anyway
    def __init__(self,L_1,L_2,g, M):

        self.L_1 = L_1 #meter
        self.L_2 = L_2
        self.M = M #kg
        self.g = g #m/s^2

        self._theta1 = None
        self._theta2 = None
        self._omega1 = None
        self._omega2 = None
        self._t = None

    def __call__(self,t,y):
        theta1,omega1,theta2,omega2 = y
        deltaTheta = theta2-theta1
        dtheta1_dt = omega1
        domega1_dt = (self.L_1*omega1**2*np.sin(deltaTheta)*np.cos(deltaTheta)+self.g*np.sin(theta2)*np.cos(deltaTheta)+self.L_2*omega2**2*np.sin(deltaTheta)-2*
        dtheta2_dt = omega2
        domega2_dt = (-self.L_2*omega2**2*np.sin(deltaTheta)*np.cos(deltaTheta)+2*self.g*np.sin(theta1)*np.cos(deltaTheta)-2*self.L_1*omega1**2*np.sin(deltaThet
        return dtheta1_dt, domega1_dt, dtheta2_dt, domega2_dt
```

Vi har mye større likninger her som skal implementeres i call metoden. Dette er uttrykk for posisjonen/bevegelsen av pendelen.

## Part 4 Animating the pendulum

```
def create_animation(self):
    # Create empty figure
    fig = plt.figure()

    # Configure figure
    plt.axis('equal')
    plt.axis('off')
    plt.axis((-3, 3, -3, 3))

    # Make an "empty" plot object to be updated throughout the animation
    self.pendulums, = plt.plot([], [], 'o-', lw=2)

    # Call FuncAnimation
    self.animation = animation.FuncAnimation(fig,
                                             self._next_frame,
                                             frames=range(len(self.x1)),
                                             repeat=None,
                                             interval=1000*self.dt,
                                             blit=True)
def _next_frame(self, i):
    self.pendulums.set_data((0, self.x1[i], self.x2[i]),
                            (0, self.y1[i], self.y2[i]))
    return self.pendulums,

def show_animation(self):
    anim = self.create_animation()
    plt.show()

#This part is incomplete and creates error
def save_animation(self,videoname):
    self.videoname = str(videoname)
    anim = self.create_animation()
    return anim.save(videoname, fps=60)
```

# Fungerer det? Testing

## Tester exp decay

```python
from exp_decay import ExponentialDecay
import pytest

def test_expdecay_val():

    u = 3.2
    a = 0.4
    expected = 1.28
    instans = ExponentialDecay(a)
    actual = instans(1,u)
    print(actual)
    tol = 1e-35
    diff = actual-expected
    assert diff<tol

def test_expdecay_num():
    with pytest.raises(ValueError):
        instans = ExponentialDecay(-1)
```

## Tester enkel pendel

```python
import numpy as np
from pendulum import Pendulum
import pytest
def test_pendulum():
    tol = 1e-16
    theta = np.pi/6
    omega = 0.15
    L = 2.7
    M = 1
    g = 9.81
    inst = Pendulum(L,M,g)
    th,om = inst(1,(theta,omega))

    expected = (-g/L)*np.sin(theta)

    difference = om-expected
    assert difference<tol

def test_ZeroTestPendulum():
    tol = 1e-16
    theta = 0
    omega = 0
    L = 2.7
    M = 1
    g = 9.81
    inst = Pendulum(L,M,g)
    th,om = inst(1,(theta,omega))

    expected = 0

    diff1 = om-expected
    diff2 = th-expected
    assert diff1<tol
    assert diff2<tol

def test_pendulumclass():
    with pytest.raises(Exception):
        instans = Pendulum()

def secondZerotestPendulum():
    theta = 0
    omega = 0
    inst = Pendulum()
    th,om = inst(1,(theta,omega))
    print(th,om)
```

## Tester dobbel pendel

```python
import pytest
from double_pendulum import DoublePendulum
import numpy as np
@pytest.mark.parametrize(
    "theta1, theta2, expected",
    [
        (  0,   0,           0),
        (  0, 0.5,  3.386187037),
        (0.5,   0, -7.678514423),
        (0.5, 0.5, -4.703164534),
    ]
)
def test_domega1_dt(theta1, theta2, expected):
    dp = DoublePendulum(L_1=1,L_2 =1, g = 9.81, M =1)
    t = 0
    y = (theta1, 0.25, theta2, 0.15)
    dtheta1_dt, domega1_dt, _, _ = dp(t, y)
    assert np.isclose(dtheta1_dt, 0.25)
    assert np.isclose(domega1_dt, expected)

@pytest.mark.parametrize(
    "theta1, theta2, expected",
    [
        (  0,   0,         0.0),
        (  0, 0.5, -7.704787325),
        (0.5,   0,  6.768494455),
        (0.5, 0.5,         0.0),
    ],
)
def test_domega2_dt(theta1, theta2, expected):
    dp = DoublePendulum(L_1=1,L_2 =1,g = 9.81, M =1)
    t = 0
    y = (theta1, 0.25, theta2, 0.15)
    _, _, dtheta2_dt, domega2_dt = dp(t, y)
    assert np.isclose(dtheta2_dt, 0.15)
    assert np.isclose(domega2_dt, expected)


#unit tests
def test_1doublependulumclass():
    with pytest.raises(Exception):
        instans = DoublePendulum()

def test_2doublependulumclass():
    with pytest.raises(TypeError):
        inst = DoublePendulum()
```

```
arminalaei@Armins-MacBook-Pro-2 H21_project1_arminal % pytest test_double_pendulum.py
============================== test session starts ==============================
platform darwin -- Python 3.9.1, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /Users/arminalaei/Documents/Fysikk/IN1910/H21_project1_arminal
collected 10 items

test_double_pendulum.py ..........                                        [100%]

============================== 10 passed in 0.58s ===============================
arminalaei@Armins-MacBook-Pro-2 H21_project1_arminal %
```

```
arminalaei@Armins-MacBook-Pro-2 H21_project1_arminal % pytest test_exp_decay.py
============================== test session starts ==============================
platform darwin -- Python 3.9.1, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /Users/arminalaei/Documents/Fysikk/IN1910/H21_project1_arminal
collected 2 items

test_exp_decay.py ..                                                      [100%]

=============================== 2 passed in 0.53s ===============================
arminalaei@Armins-MacBook-Pro-2 H21_project1_arminal %
```

```
arminalaei@Armins-MacBook-Pro-2 H21_project1_arminal % pytest test_pendulum.py
============================== test session starts ==============================
platform darwin -- Python 3.9.1, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /Users/arminalaei/Documents/Fysikk/IN1910/H21_project1_arminal
collected 3 items

test_pendulum.py ...                                                      [100%]

=============================== 3 passed in 7.64s ===============================
```