Oppg1

a) Bruker numeric division.

Er også vist I koden nederst I dokumentet.

b) Bruker numerisk integrasjon. Med algoritmen

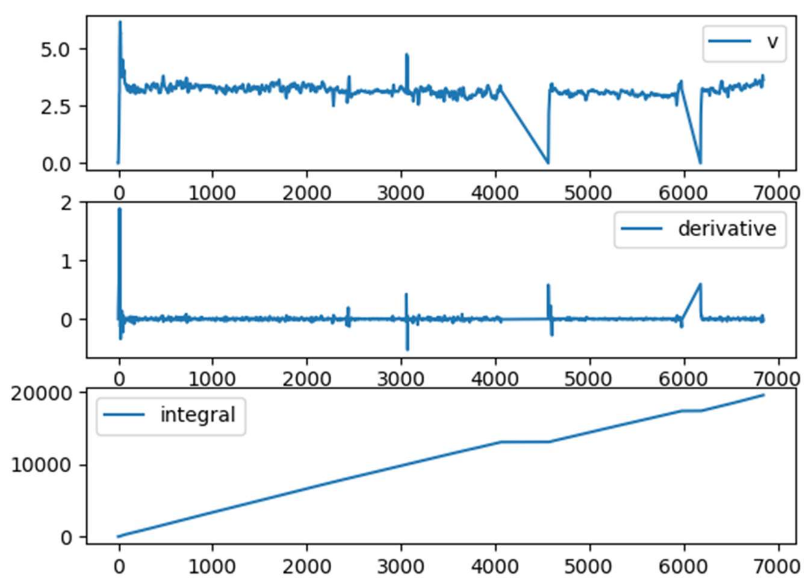
for $i = 1, \dots, \mathrm{len}(Y)$

$$dx = x[i] + x[i-1]$$

$$Y_{x+1} = Y_i \cdot dx + Y_{i-1}$$

Der Y[i] er integrasjonen fra 0 til i av Y. også vist i koden.

c) Framgangsmåte i koden.

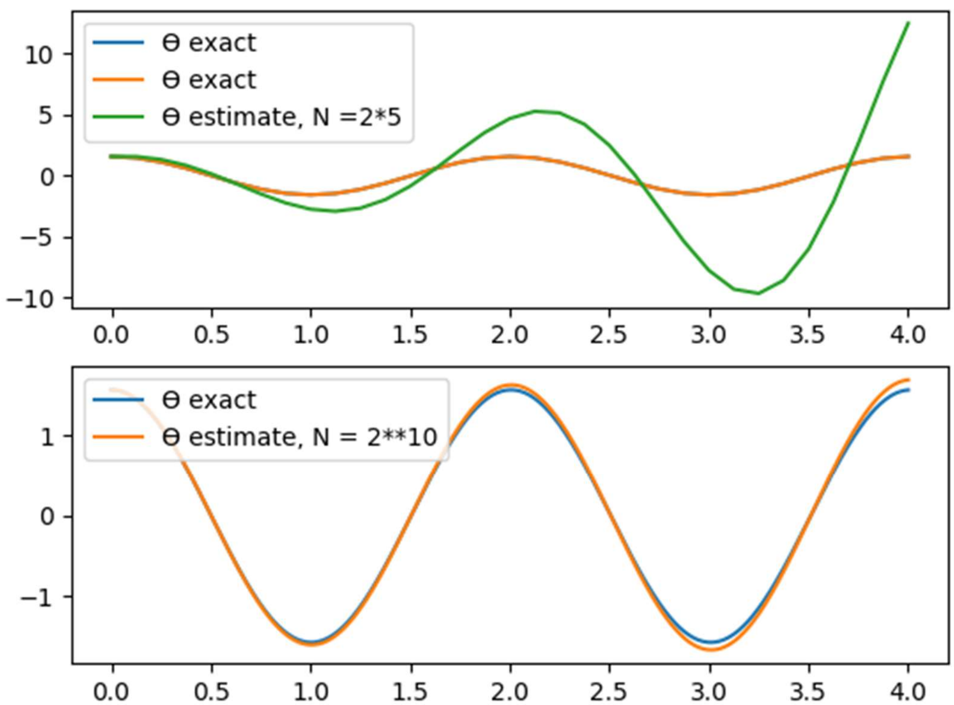


Integralet viser lengden, derivate viser akselerasjonen, og V viser farten.

Framgangsmåte ligger i koden.

C) I python koden

d)



Ser at estimatet blir mer nøyaktig når vi øker N, og at ved liten N så blir estimatetr mer og mer usikkert jo lengere fra x=0 vi kommer.

e) Estimatet er

N = 16

err[1] = 0.00011756377981275712

err[103] = 0.0038099404835139072

err[205] = 0.019454896734367066

err[307] = 0.030003221985847484

err[409] = 0.013649685114557941

err[511] = 0.0611889219864572

err[613] = 0.026145718107934424

err[715] = 0.06684992331497353

err[817] = 0.08377880498442503

err[919] = 0.026552753305578047

err[1021] = 0.12430267382042826

N = 32

err[1] = 0.00011756377981275712

err[103] = 0.0038099404835139072

err[205] = 0.019454896734367066

err[307] = 0.030003221985847484

err[409] = 0.013649685114557941

err[511] = 0.0611889219864572

err[613] = 0.026145718107934424

err[715] = 0.06684992331497353

err[817] = 0.08377880498442503

err[919] = 0.026552753305578047

err[1021] = 0.12430267382042826

N = 64

err[1] = 0.00011756377981275712

err[103] = 0.0038099404835139072

err[205] = 0.019454896734367066

err[307] = 0.030003221985847484

err[409] = 0.013649685114557941

err[511] = 0.0611889219864572

err[613] = 0.026145718107934424

err[715] = 0.06684992331497353

err[817] = 0.08377880498442503

err[919] = 0.026552753305578047

err[1021] = 0.12430267382042826

N = 128

err[1] = 0.00011756377981275712

err[103] = 0.0038099404835139072

err[205] = 0.019454896734367066

err[307] = 0.030003221985847484

err[409] = 0.013649685114557941

err[511] = 0.0611889219864572

err[613] = 0.026145718107934424

err[715] = 0.06684992331497353

err[817] = 0.08377880498442503

err[919] = 0.026552753305578047

err[1021] = 0.12430267382042826

N = 256

err[1] = 0.00011756377981275712

err[103] = 0.0038099404835139072

err[205] = 0.019454896734367066

err[307] = 0.030003221985847484

err[409] = 0.013649685114557941

err[511] = 0.0611889219864572

err[613] = 0.026145718107934424

err[715] = 0.06684992331497353

err[817] = 0.08377880498442503

err[919] = 0.026552753305578047

err[1021] = 0.12430267382042826

N = 512

err[1] = 0.00011756377981275712

err[103] = 0.0038099404835139072

err[205] = 0.019454896734367066

err[307] = 0.030003221985847484

err[409] = 0.013649685114557941

err[511] = 0.0611889219864572

err[613] = 0.026145718107934424

err[715] = 0.06684992331497353

err[817] = 0.08377880498442503

err[919] = 0.026552753305578047

err[1021] = 0.12430267382042826

N = 1024

err[1] = 0.00011756377981275712

err[103] = 0.0038099404835139072

err[205] = 0.019454896734367066

err[307] = 0.030003221985847484

err[409] = 0.013649685114557941

err[511] = 0.0611889219864572

err[613] = 0.026145718107934424

err[715] = 0.06684992331497353

err[817] = 0.08377880498442503

err[919] = 0.026552753305578047

err[1021] = 0.12430267382042826

f) Er i koden
g) Er i koden

Koden

```
import numpy as np

import matplotlib.pyplot as plt

#c)


def lin_pendel_euler(v0, theta0, g, L, N, T, h=None):
    if h is None:
        h = T/N
    v, theta = [0]*(N+1), [0]*(N+1)
    v[0], theta[0] = v0, theta0

    for k in range(N):
        v[k+1] = v[k] - g*h*theta[k]
        theta[k+1] = theta[k] + h*(v[k]/L)
    return v, theta




#d)
def lin_pendel(t, v0, theta0, g, L):
    return theta0 * np.cos(np.sqrt(g/L)*t) + v0 * np.sin(np.sqrt(g/L)*t)
```

```
g, L, v0, theta0, T, N1 = 9.81, 1, 0, np.pi/2, 4, 2**5
v_hat, theta_hat = lin_pendel_euler(v0, theta0, g, L, N1, T)

t1 = np.linspace(0,T,N1+1) #[h*k for k in range(N)]
theta1 = lin_pendel(t1, v0, theta0, g, L)

plt.subplot(2, 1, 1)
plt.plot(t1,theta1, label='Θ exact')
plt.plot(t1,theta_hat, label='Θ estimate, N =2*5')
plt.legend()

plt.subplot(2, 1, 2)
N2 = 2**10
v_hat, theta_hat = lin_pendel_euler(v0, theta0, g, L, N2, T)
t2 = np.linspace(0,T,N2+1) #[h*k for k in range(N)]
theta2 = lin_pendel(t2, v0, theta0, g, L)

plt.plot(t2,theta2, label='Θ exact')
plt.plot(t2,theta_hat, label='Θ estimate, N = 2**10')
plt.legend()

plt.show()



#e)

def epsilon(N, h=None):
    t = np.linspace(0,T,N+1)
    theta = lin_pendel(t, v0, theta0, g, L)
    theta_hat = np.asarray(lin_pendel_euler(v0, theta0, g, L, N, T, h)[-1])
```

```
    err = np.absolute(theta-theta_hat)
    return err

n = [2**i for i in range(4,10+1)]

for N in n:
    err = epsilon(2**10)
    print('''---------------
    N = {}'''.format(N))
    for i in range(1, len(err), int(len(err)/10)):
        print('err[{}] = {}'.format(i, err[i]))



#f)

def p(epsilon):
    N = 2**4
    h2 = (T/N)/2
    h1 = T/N
    return (np.log(epsilon(N)/epsilon(N)))/np.log(h1/h2)



print(p(epsilon))

#g
def pendel_euler(v0, theta0, g, L, N, h):
    v = theta =np.zeros(N+1)
    v[0], theta[0] = v0, theta0
    for k in range(N):
        v[k+1] = v[k] - g*h*np.sin(theta[k])
        theta[k+1] = theta[k] + h * v[k]/L
```

```
    return v, theta

h1 = T/N1
h2 = T/N2
v_pen1, theta_pen1 = pendel_euler(v0, theta0, g, L, N1, h1)
v_pen2, theta_pen2 = pendel_euler(v0, theta0, g, L, N2, h2)



plt.subplot(2, 1, 1)
plt.plot(t1,theta1, label='ϴ exact')
plt.plot(t1,theta_hat, label='ϴ estimate, N =2*5')
plt.plot(t1,theta_pen1, label = 'ϴ theta_pen, N = 2*5')
plt.legend()

plt.subplot(2, 1, 2)
N2 = 2**10
v_hat2, theta_hat2 = lin_pendel_euler(v0, theta0, g, L, N2, T)

theta = lin_pendel(t2, v0, theta0, g, L)

plt.plot(t2,theta2, label='ϴ exact')
plt.plot(t2,theta_hat2, label='ϴ estimate, N = 2**10')
plt.plot(t2,theta_pen2, label = 'ϴ theta_pen, N = 2*10')
plt.legend()
plt.show()
```

Kjøreeksempel

```
---------------
    N = 16
err[1] = 0.00011756377981275712
err[103] = 0.0038099404835139072
err[205] = 0.019454896734367066
err[307] = 0.030003221985847484
err[409] = 0.013649685114557941
err[511] = 0.0611889219864572
err[613] = 0.026145718107934424
```

```
err[715] = 0.06684992331497353
err[817] = 0.08377880498442503
err[919] = 0.026552753305578047
err[1021] = 0.12430267382042826
---------------
    N = 32
err[1] = 0.00011756377981275712
err[103] = 0.0038099404835139072
err[205] = 0.019454896734367066
err[307] = 0.030003221985847484
err[409] = 0.013649685114557941
err[511] = 0.0611889219864572
err[613] = 0.026145718107934424
err[715] = 0.06684992331497353
err[817] = 0.08377880498442503
err[919] = 0.026552753305578047
err[1021] = 0.12430267382042826
---------------
    N = 64
err[1] = 0.00011756377981275712
err[103] = 0.0038099404835139072
err[205] = 0.019454896734367066
err[307] = 0.030003221985847484
err[409] = 0.013649685114557941
err[511] = 0.0611889219864572
err[613] = 0.026145718107934424
err[715] = 0.06684992331497353
err[817] = 0.08377880498442503
err[919] = 0.026552753305578047
err[1021] = 0.12430267382042826
---------------
    N = 128
err[1] = 0.00011756377981275712
err[103] = 0.0038099404835139072
err[205] = 0.019454896734367066
err[307] = 0.030003221985847484
err[409] = 0.013649685114557941
err[511] = 0.0611889219864572
err[613] = 0.026145718107934424
err[715] = 0.06684992331497353
err[817] = 0.08377880498442503
err[919] = 0.026552753305578047
err[1021] = 0.12430267382042826
---------------
    N = 256
err[1] = 0.00011756377981275712
err[103] = 0.0038099404835139072
err[205] = 0.019454896734367066
err[307] = 0.030003221985847484
err[409] = 0.013649685114557941
err[511] = 0.0611889219864572
err[613] = 0.026145718107934424
err[715] = 0.06684992331497353
err[817] = 0.08377880498442503
err[919] = 0.026552753305578047
err[1021] = 0.12430267382042826
---------------
    N = 512
err[1] = 0.00011756377981275712
err[103] = 0.0038099404835139072
err[205] = 0.019454896734367066
err[307] = 0.030003221985847484
err[409] = 0.013649685114557941
err[511] = 0.0611889219864572
err[613] = 0.026145718107934424
err[715] = 0.06684992331497353
err[817] = 0.08377880498442503
err[919] = 0.026552753305578047
err[1021] = 0.12430267382042826
---------------
```

```
    N = 1024
err[1] = 0.00011756377981275712
err[103] = 0.0038099404835139072
err[205] = 0.019454896734367066
err[307] = 0.030003221985847484
err[409] = 0.013649685114557941
err[511] = 0.0611889219864572
err[613] = 0.026145718107934424
err[715] = 0.06684992331497353
err[817] = 0.08377880498442503
err[919] = 0.026552753305578047
err[1021] = 0.12430267382042826
[nan  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
D:\OneDrive - Universitetet i Oslo\UIO\MAT-INF1100\Oblig2\oppg2.py:71: RuntimeWarning: invalid v
alue encountered in true_divide
  return (np.log(epsilon(N)/epsilon(N)))/np.log(h1/h2)
Traceback (most recent call last):
  File "D:\OneDrive - Universitetet i Oslo\UIO\MAT-INF1100\Oblig2\oppg2.py", line 93, in <module
>
    plt.plot(t1,theta_hat, label='\u03f4 estimate, N =2*5')
  File "C:\Users\SKJSA\AppData\Local\Programs\Python\Python37\lib\site-packages\matplotlib\pyplo
t.py", line 2749, in plot
    *args, scalex=scalex, scaley=scaley, data=data, **kwargs)
  File "C:\Users\SKJSA\AppData\Local\Programs\Python\Python37\lib\site-packages\matplotlib\__ini
t__.py", line 1785, in inner
    return func(ax, *args, **kwargs)
  File "C:\Users\SKJSA\AppData\Local\Programs\Python\Python37\lib\site-packages\matplotlib\axes\
_axes.py", line 1604, in plot
    for line in self._get_lines(*args, **kwargs):
  File "C:\Users\SKJSA\AppData\Local\Programs\Python\Python37\lib\site-packages\matplotlib\axes\
_base.py", line 393, in _grab_next_args
    yield from self._plot_args(this, kwargs)
  File "C:\Users\SKJSA\AppData\Local\Programs\Python\Python37\lib\site-packages\matplotlib\axes\
_base.py", line 370, in _plot_args
    x, y = self._xy_from_xy(x, y)
  File "C:\Users\SKJSA\AppData\Local\Programs\Python\Python37\lib\site-packages\matplotlib\axes\
_base.py", line 231, in _xy_from_xy
    "have shapes {} and {}".format(x.shape, y.shape))
ValueError: x and y must have same first dimension, but have shapes (33,) and (1025,)
[Finished in 2.176s]
```