# Oppg1

## 1a: Gi en algoritme for å beregne en tilnærming til objektets aksellerasjon a(t) = v 0 (t) ut fra de beregnede verdiene (ti , vi) av farten.

Siden vi har hastighet gitt av tid. Må vi derivere for å finne akselerasjonen. Dette gjøres ved å bruke numerisk derivasjon. Siden vi har data fra punkt data og ikke en kontinuerlig funksjon, bruker vi en steglende gidd ved t\_i – t\_i+1. Da er algoritmen:

def aksellerasjon (x,y):

dy = [(y[i+1] - y[i])/ abs(x[i+1] - x[i]) for i in range(len(y)-1)]

return(dy)

Som da returnerer den deriverte av y, som i vår oppgave er v.

Eksempel:

t = list(range(0,10,2))

v = [2,4,8,12,24]

print(t[1:],aksellerasjon(t,v))

python .\1a.py

[2, 4, 6, 8] [1.0, 2.0, 2.0, 6.0]

## 1b: Gi en algoritme for å beregne en tilnærming til objektets avstand s(t) fra startpunktet ut fra de beregnede verdiene når v(t) = s 0 (t) og s(t0) = 0.

For å finne avstanden fra startpunktet bruker jeg numerisk integrasjon. Som i 1a må jeg bruke integrasjonen over punkter og ikke en funksjon. Det gjør jeg med å regne ut dx for hver element i y. Det gir meg algoritmen:

def avstand(x,y):

Y = [0]

for i in range(1,len(y)):

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

Y.append(y[i]\*dx + Y[-1])

return Y

Eksempel:

t = list(range(0,10,2))

v = [2,4,8,12,24]

print(t, avstand(t,v))

python .\1b.py

[0, 2, 4, 6, 8] [0, 8, 24, 48, 96]

## 1c: Last ned fila running.txt og kjør denne koden, og bruk algoritmene fra a) og b) til å lage to plott: Ett der du plotter objektets akselerasjon mot tid, og ett der du plotter objektets avstand fra startpunktet mot tid.

from Oppg1a import \*

from Oppg1b import \*

import numpy as np

import matplotlib.pyplot as plt

t = []

v = []

infile = open('running.txt','r')

for line in infile:

tnext, vnext = line.strip().split(',')

t.append(float(tnext))

v.append(float(vnext))

infile.close()

V = np.asarray(avstand(t, v))

dv = np.asarray(aksellerasjon(t, v))

if \_\_name\_\_ == "\_\_main\_\_":

plt.subplot(211)

plt.plot(t[1:],dv,label='aksellerasjon')

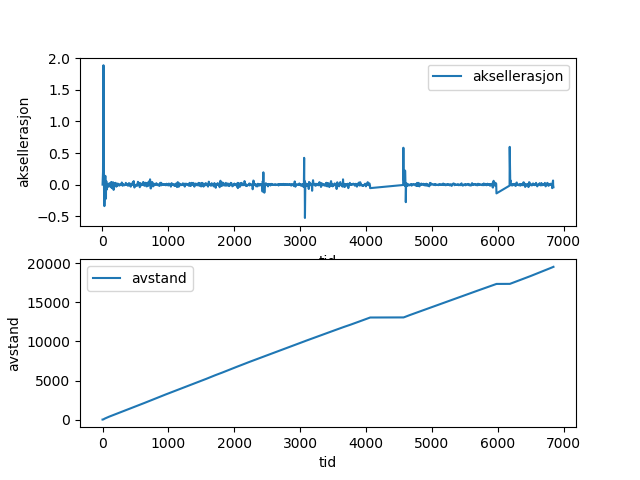
plt.legend()

plt.subplot(212)

plt.plot(t, V, label='avstand')

plt.legend()

plt.show()



# Oppg2

## 2a:

## 2b:

## 2c: Skriv en Python-funksjon lin\_pendel\_euler som med funksjonskallet v, theta = lin\_pendel\_euler(v0, theta0, g, L, N, h).

import numpy as np

import matplotlib.pyplot as plt

def lin\_pendel\_euler(v0, theta0, g, L, N, h=1e-14):

T = h\*N #siden vi har 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

## 2d: Plott den numeriske løsningen for vinkelutslaget som en funksjon av tid sammen med den eksakte løsningen du fant i b). Kommenter forskjellene.

from Oppg2c import \*

import numpy as np

import matplotlib.pyplot as plt

def lin\_pendel(t, v0, theta0, g, L):

return theta0 \* np.cos(np.sqrt(g/L)\*t) + v0/np.sqrt(g\*L) \* np.sin(np.sqrt(g/L)\*t) #løsning fra 2b

g, L, v0, theta0 = 9.81, 1, 0, np.pi/2

N = [2\*\*5, 2\*\*10]

T = 4

h = [T/n for n in N]

t = [np.linspace(0,T,n+1) for n in N]

plt.subplot(2, 1, 1)

v\_hat, theta\_hat = lin\_pendel\_euler(v0, theta0, g, L, N[0], h[0])

theta = lin\_pendel(t[0], v0, theta0, g, L)

plt.plot(t[0], theta, label='ϴ exact')

plt.plot(t[0], theta\_hat, label='ϴ estimate, N ={}'.format(N[0]))

plt.legend()

plt.subplot(2, 1, 2)

v\_hat, theta\_hat = lin\_pendel\_euler(v0, theta0, g, L, N[1], h[1])

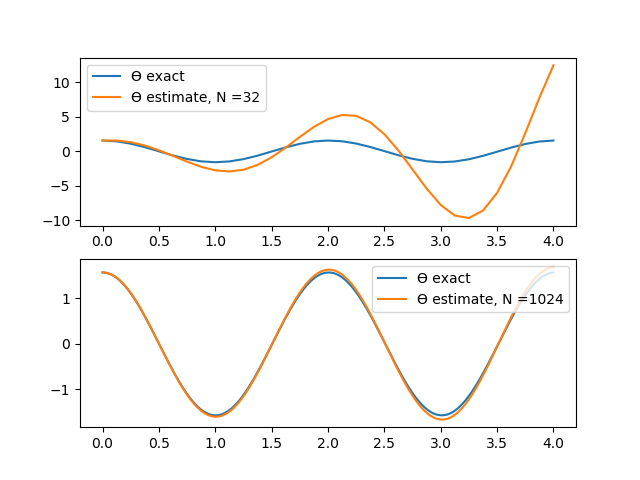
theta = lin\_pendel(t[1], v0, theta0, g, L)

plt.plot(t[1], theta, label='ϴ exact')

plt.plot(t[1], theta\_hat, label='ϴ estimate, N ={}'.format(N[1]))

plt.legend()

plt.show()



Ser at estimatet til theta har en økende svingning. Noe som ikke er synlig i den eksakte theta. Da en pendel ikke kan svinge bredere enn i starten. Dermed er ikke theta et perfekt estimat.

## 2e: beregne feilen i Eulers metode

from Oppg2c import \*

from Oppg2d import \*

import numpy as np

import matplotlib.pyplot as plt

class epsilon:

def \_\_init\_\_(self,f, f\_hat,v0, theta0, g, L, T):

self.f = f

self.f\_hat = f\_hat

self.v0, self.theta0, self.L= v0, theta0, L

self.T = T

def \_\_call\_\_(self, h, N):

#feilestiamtet

v0, theta0, L = self.v0, self.theta0, self.L

T = self.T

t = np.linspace(0,T,int(N+1))

f = self.f(t, v0, theta0, g, L)

f\_hat = self.f\_hat(v0, theta0, g, L, N, h)[1]

return abs(f[-1]-f\_hat[-1])

if \_\_name\_\_ == "\_\_main\_\_":

N = [2\*\*i for i in range(4,11)]

T = 4

h = [T/n for n in N]

g, L, v0, theta0 = 9.81, 1, 0, np.pi/2

print('''-------------------------

| N h e(p) |''')

for i in range(len(N)):

e = epsilon(lin\_pendel,lin\_pendel\_euler,v0, theta0, g, L, T)

print('|{:.2e} {:.4f} {:7.4f}|'.format(N[i], h[i], e(h[i], N[i]) ))

run Oppg2e.py

-------------------------

| N h e(p) |

|1.60e+01 0.2500 27.3939|

|3.20e+01 0.1250 10.9065|

|6.40e+01 0.0625 3.5644|

|1.28e+02 0.0312 1.3131|

|2.56e+02 0.0156 0.5614|

|5.12e+02 0.0078 0.2598|

|1.02e+03 0.0039 0.1250|

For å beregne feilestiamtet satte jeg opp en klasse med en init for å ta inn konstantene. Så en call metode for å kalle funksjonen. Så avslutter med en print slik at vi får error estimatet i en tabell.

## 2f: Bruk tallene du fant i oppgave e) til å estimere konvergensraten. . Hva tror du konvergensraten til Eulers metode er?

from Oppg2c import \*

from Oppg2d import \*

from Oppg2e import \*

import numpy as np

import matplotlib.pyplot as plt

class konvergens(epsilon):

def p(self, h1, N):

e = self.\_\_call\_\_

h2 = h1/2

return np.log(e(h1, N)/e(h2, N\*2))/np.log(h1/h2)

if \_\_name\_\_ == "\_\_main\_\_":

N = [2\*\*i for i in range(4,11)]

T = 4

h = [T/n for n in N]

print('''-----------------------------------

| N h e(p) p |''')

for i in range(len(N)):

e = konvergens(lin\_pendel,lin\_pendel\_euler,v0, theta0, g, L, T)

p = e.p(h[i], N[i])

print('| {:.2e}{:8.4f}{:8.4f}{:8.4f}|'.format(N[i], h[i], e(h[i], N[i]), p))

Definerer en ny classe som arver fra epsilon. Definerer en ny metode p som kalkulerer p med samme init som epsilion. Så gjør et tilsvarende plott som i 2e, med p i tilleg.

Vi får da ut tabellen:

run Oppg2f

-----------------------------------

| N h e(p) p |

| 1.60e+01 0.2500 27.3939 1.3287|

| 3.20e+01 0.1250 10.9065 1.6135|

| 6.40e+01 0.0625 3.5644 1.4406|

| 1.28e+02 0.0312 1.3131 1.2259|

| 2.56e+02 0.0156 0.5614 1.1119|

| 5.12e+02 0.0078 0.2598 1.0555|

| 1.02e+03 0.0039 0.1250 1.0276|

Ser ut til at p går mot 1.

## 2g: Skriv en Python-funksjon pendel\_euler. Plott θ som en funksjon av tid. Inkluder de to løsningene fra b) og c) i plottet. Kommenter forskjellene.

from Oppg2f import \*

import matplotlib.pyplot as plt

import numpy as np

def pendel\_euler(v0, theta0, g, L, N, h):

v = np.zeros(N+1)

theta = np.zeros(N+1)

v[0], theta[0] = v0, theta0

for k in range(N):

v[k+1] = v[k] - g\*np.sin(theta[k])\*h

theta[k+1] = theta[k] + h\* (1/L \* v[k])

return v, theta

g, L, v0, theta0 = 9.81, 1, 0, np.pi/2

T = 4

N = 2\*\*10

h = T/N

t = np.linspace(0,T,N+1)

v\_pendel\_euler, theta\_pendel\_euler = pendel\_euler(v0, theta0, g, L, N, h)

theta\_lin\_pendel = lin\_pendel(t, v0, theta0, g, L)

v\_lin\_pendel\_euler, theta\_lin\_pendel\_euler = lin\_pendel\_euler(v0, theta0, g, L, N, h)

plt.plot(t,theta\_pendel\_euler, label='theta\_pendel\_euler')

plt.plot(t,theta\_lin\_pendel, label='theta\_lin\_pendel')

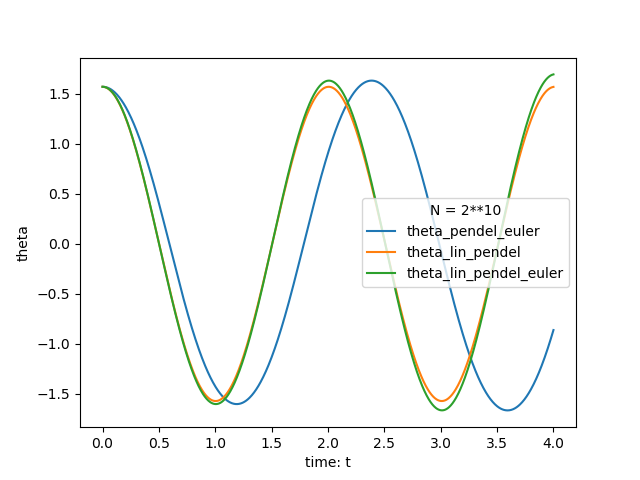
plt.plot(t,theta\_lin\_pendel\_euler, label='theta\_lin\_pendel\_euler')

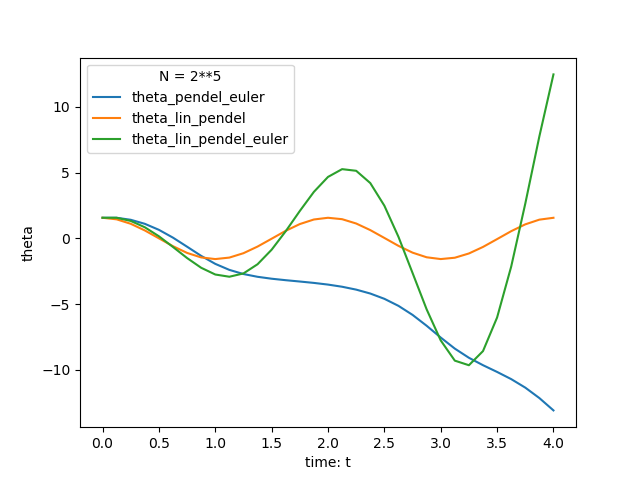
plt.xlabel('time: t')

plt.ylabel('theta')

plt.legend(title ='N = 2\*\*10')

plt.show()



VI ser at theta\_pendel\_euler raskt får en større feil en det theta\_lin\_pendel har. Hvis vi ser på en mindre N så får vi. 

Der vi ser at theta\_lin\_pendel\_euler raskt blir feil men føler fortsatt theta\_lin pendel. Mens theta\_pendel\_euler er mindre feil men går i feil retning. Dermed mener jeg at theta\_lin\_pendel\_euløer er det beste estimatet.