FYS2130 Oblig 3

Daniel Heinesen, daniehei

12. februar 2017

Diskusjon

1)

Eulers metode er av første orden(feil av orden Δt), og bruker bare ett punk til å tilnærme det neste punktet. Dette gjør at denne metoden er svært unøyaktig. Dette gjelder spesielt periodiske systemer, hvor Eulers metode ikke bevarer energi. Dette fører ofte til at man før enormt overdrevede resultater om man bruker denne metoden på slike systemer.

Runge Kutta 4(RK4) er en metode i Runge Kutta-familien. Denne familien bruker numerisk integralsjon til å løse differensiallikninger. RK4 bruker Simpsons regel for å løse disse integralene. RK4 bruke 4 punkter for å finne neste punkt. Dette betyr at den er veldig nøyaktig, og har en feil av orden Δt^4. Den bevarer også energi i periodiske systemer, noe som gjør den mye mer egnet til våre formål.

Regneoppgaver

4)

a)

Verdiene som brukes i denne oppgaven er $m = 0.1\text{kg}$, $k = 10\text{N/m}$ og $b = 0.1\text{kg/s}$. Med disse tallene vet vi at

$$\gamma = \frac{b}{2m} = \frac{1}{2}, \qquad \omega = \sqrt{\frac{k}{m}} = 10$$

Dette betyr at

$$\gamma < \omega$$

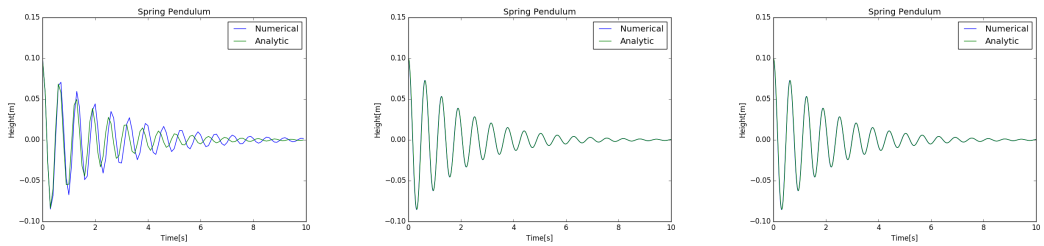
Og vi har en underkritisk demping. Vi vet da at den analytiske løsningen er

$$z(t) = e^{-\gamma t} A \cos(\omega' t + \phi)$$

hvor $\omega' = \sqrt{\omega^2 - \gamma^2}$, $A = x_0 = 0.1\text{m}$, og $\phi = 0$.

Koden finnes her 1(finnes også i filen *pendulumRK4.py*)

Jeg testet så programmet med 3 forskjellige tidssteg dt, som er vist i figuren under



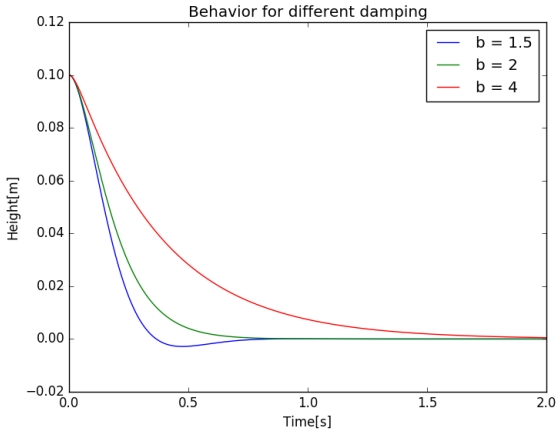
(a) $dt = 0.1$ (b) $dt = 0.01$ (c) $dt = 0.001$

Figur 1: Plot med 3 forskjellige tidssteg.

Vi ser at $dt = 0.1$ er altfor smått og det er en klar forskjell mellom den numeriske og den analytiske løsningen. Forskjellen mellom $dt = 0.01$ og 0.001 er derimot ikke så stor, og vi kan derfor si at den numerisk løsningen har stabilisert seg her. For å få den best mulige løsningen har jeg valgt å bruke $dt = 0.001$ videre.

b)

Vi kan så se på hvordan systemet oppfører seg med overdemping, underdemping og ved kritisk demping. Jeg gjorde dette ved å justere verdiene for b



Figur 2: Figuren viser hvordan fjærpendelen oppfører seg for forskjellige verdier av dempingsfaktoren b

Verdiene til b ble valgt ved å se på ω og γ. For å finne kritisk demping satt jeg

$$\omega = \gamma \Rightarrow b = 2m\sqrt{\frac{k}{m}} = 2$$

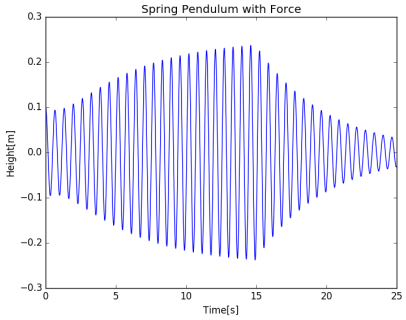
For overkritisk demping ble må $b > 2$ og for underkritiskdeming må $b < 2$. Disse verdiene ble valg slik at plottene var lette å skille fra hverander.

c)

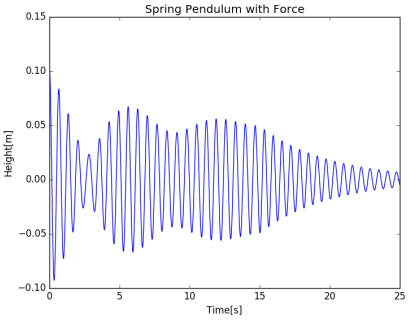
Legger vi nå til en påtrykt kraft av formen

$$F_0 \cos(\omega_F t)$$

Hvor $F_0 = 0.1$N. Forskjellige verdier av ω_F vil gi forskjellig oppførsel. Om $\omega_F = \omega$, vil systemet være i resonans, og ampituden vil øke til en A_{max}. Er dette ikke tilfelle vil størrelsen på amplituden variere periodisk. Her har jeg simulert begge tilfellene. Ca halvveis har jeg skrudd av den påtrykte kraften, slike at svigningene dempes og dør ut.



(a) $\omega_F = \omega$

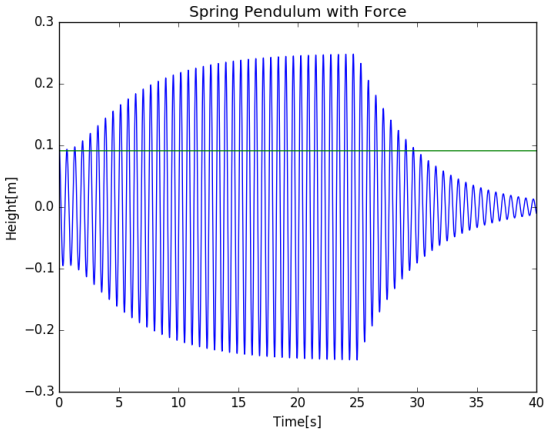


(b) $\omega_F = 0.9\omega$

Vi kan se akkurat det vi forventet, om $\omega_F = \omega$ osillerer amplituden periodisk. Når kraften skrues av etter ca 15 sek dør svigningene sakte ut.

d)

Vi kan så sjekke hva frekvensresponsen til systemet er. Vi vil sjekke når svigningen har falt til $1/e$ av maksamplituden



Figur 4: Den grønnelinjen viser hvor amplituden er $1/e$ av maksamplituden.

Er skrues den påtrykte kraften av etter 25 sek. Vi kan se at amplituden krysser den grønne linjen ved ca 30 sek. Med andre ord $\Delta t = 5$ sek. Vi kan nå finne Q-verdien

$$Q = \omega \Delta t = \sqrt{\frac{b}{m}} \Delta t = \sqrt{\frac{0.04}{0.1}} 5 = 3.16$$

7)

Kode:

Kode for 4a)

Listing 1: Kode for 4a

```
import numpy as np
```

```
import matplotlib.pyplot as plt



def RK4(T,dt,x0,v0,a):

    dim = len(x0)
    timeSteps = int(float(T)/dt)
    t = np.zeros(timeSteps)
    x = np.zeros((timeSteps,dim))
    v = np.zeros((timeSteps,dim))
    x[0] = x0
    v[0] = v0


    for i in range(0,int(timeSteps)-1):

        v1 = v[i]
        a1 = a(t[i],x[i],v[i])

        v2 = v[i] + a1*dt/2.0
        a2 = a(t[i] + 0.5*dt, x[i] + v2*dt*0.5,v2)

        v3 = v[i] + a2*dt/2.0
        a3 = a(t[i] + 0.5*dt, x[i] + v3*dt*0.5,v3)

        v4 = v[i] + a3*dt
        a4 = a(t[i] + dt, x[i] + v3*dt,v4)

        aMid = 1.0/6.0*(a1 + 2*a2 + 2*a3 + a4)
        vMid = 1.0/6.0*(v1 + 2*v2 + 2*v3 + v4)


        t[i+1] = t[i] + dt
        x[i+1] = x[i] + vMid*dt
        v[i+1] = v[i] + aMid*dt

    return t,x,v



def a(t,x,v):
    k = 10

    L = 0
    m = .1
    omega = 12
    V_0 = 0.1
    F_trykk = 0#V_0*np.sin(omega*t)
    b = 0.1
    F = k*(L-x) - b*v
    return (F+F_trykk)/m


def analytisk(t):
    k = 10
    m = .1
    b = .1
```

```
    omega = np.sqrt(k/m)
    gamma = b/(2*m)
    omega2 = np.sqrt(omega**2-gamma**2)
    A = .1
    phi = -.1


    return np.exp(-gamma*t)*A*np.cos(omega2*t+phi)



T = 10
dt = 1e-3
x0 = np.array([0,.1])
v0 = np.array([0,0])

b = 1.5

t1,x1,v1 = RK4(T,dt,x0,v0,a)


plt.plot(t1,x1[:,1])
plt.plot(t1,analytisk(t1))
plt.legend(["Numerical", "Analytic"])
plt.xlabel("Time[s]")
plt.ylabel("Height[m]")
plt.title("Spring Pendulum")
plt.show()
```