INTRODUKTION OG SETUP FOR SIMULERING AF DYNAMISKE SYSTEMER 17.04.2016

INTRODUKTION

Fysikken love indeholder ofte krafter og acceleration og kan derfor beskrives som 2. Ordens differentialligninger. Disse er ofte svære eller umulige at løse analytisk, d.v.s. finde ligningen, men den kan ofte klares nummerisk. I denne introduktion prøver vi at arbejde os frem til at kunne gøre det.

SETUP - I KAN SPRINGE OVER OG BRUGE HJEMMESIDEN MPSTEENSTRUP.DK

Vi bruger programmeringssproget Python som er installeret på alle Mac computere. Jeg foreslår at I teamer op med folk med mac computere og ikke bruger tid på at installere python, men det er vist ikke så svært på windows computere.
Vi skal bruge en tekst editor hvor vi kan gemme ren uformateret tekst. TextEdit kan det hvis I vælger FORMAT/KONVATER TIL ALM. TEKST.
Efter I har skrevet programmet skal i gemme det som python fil med endelsen .py og ikke bruge æøå eller mellemrum.
Åben en TERMINAL, den ligger under HJÆLPEPROGRAMMER.
Naviger til jeres mappe (jeg viser hvordan)

MATEMATIKKEN BAG

Vi vil gerne bruge denne metode til at løse differentialligningern dvs ligninger hvor den venstre side er differentieret eks. $x'' = -k/m \cdot x$, hvor accelerationen afhænger af positionen som i Hooks lov.
Tænk tilbage til hvordan I definerede en differentialkvotient,

$$\frac{f(t)-f(t_0)}{\Delta t} = f'(t) \ for \ \Delta t \to 0$$

Nu ganger jeg med Δt

$$f(t) - f(t_0) = f'(t) \cdot \Delta t$$

Og ligge $f(x_0)$ til på begge sider så

$$f(t) = f(t_0) + f'(t) \cdot \Delta t$$

Vi kan altså hvis vi kender startpositionen x_0 og den afledte x' finde positionen efter Δt sekunder ved at beregne
$x_1 = x_0 + x_0' \cdot \Delta t$, og fortsætte med
$x_2 = x_1 + x_1' \cdot \Delta t$
$x_3 = x_2 + x_2' \cdot \Delta t$, osv

Denne metode kaldes Eulers metode til løsning af differentialligninger. Der findes andre og mere avancerede, men denne er fin til en start.
I programmering er sproget lidt anderledes, her kan man godt skrive,
$x = x + 1$, hvilket betyder at x variablen skal opdateres med den oprindelige x værdi plus 1.

PROGRAMMERING

1D VERTIKALT KAST

Load bibliotekerne numpy og pyplot og kald dem np pg plt.

```
import numpy as np
from matplotlib import pyplot as plt
```

Setup tid mm

```
T = 2                       # tiden simulationen vare
dt = 0.01                   # tidsskridt
t = np.arange(0,T,dt)       # en liste med tidsværdier fra 0 til T
r = np.arange(0,T,dr)       # en variabel hvor vi kan gemme
x-værdierne
                            # samme længde som t.
i = 0                       # en tællevariabel
```

Fysikken

```
m=1                         # massen
g=9.82                      # tyngdeaccelerationen
```

Startbetingelser

```
x=0                         # startposition 0
dx = 10                     # starthastighed 10m/s
ddx = 0                     # startacceleration 0
```

Løkken, hvor vi opdaterer vores x position og gemmer det i kolonnen r. Husk indrykning ellers virker det ikke,

```
while t[i]<T-dt:            # fortsæt løkken så længe tiden t er
mindre
                            # end store T
     ddx = -g               # fysikken er at accelerationen er
konstant
                            # a= -g, skrevet som ddx = -g
     dx = dx + ddx*dt            # vi bruger reglen fra matematikdelen
og
                            # opdaterer hastigheden ved at addere
```

```
                          # accelerationen, ddx, gange tidsskridtet,
                          # dt
    x = x + dx*dt         # det samme gøres for x positionen
    r = np.append(r,x)    # den nye position gennem i variablen r
    i = i+1               # tællevariablen gøres én større
```

Til sidst skal vi lave en graf,

```
plt.plot(t,r)             # plt.plot siger brug pyplot til at plotte
                          # en (t,r) graf
plt.show()                # fortæller at grafen skal vises
```

1D KAST MED FRIKTION

Det er relativt let at tilføje friktion eller luftmodstand som er relevant her. Ifølge teorien er luftmodstanden proportional med hastigheden i anden,

$$F = k \cdot v^2$$

Eller accelerationen

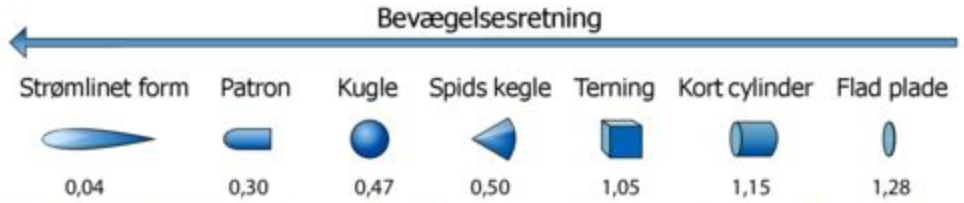
$$a = k/m \cdot v^2$$

Lidt mere udførligt er teorien

Luftmodstanden F_{luft} i en bevægelse med farten v kan med god tilnærmelse udtrykkes ved formlen

$$F_{luft} = \frac{1}{2} \cdot c_W \cdot \rho_{luft} \cdot A \cdot v^2 \qquad 6.2.8$$

Her er ρ_{luft} luftens densitet, som ifølge *Databog fysik-kemi* er $1{,}293\ \mathrm{kg/m^3}$ (ved $20\ ^\circ\mathrm{C}$ nær jordoverfladen). A er genstandens tværsnitsareal (målt vinkelret på genstandens bevægelsesretning), og c_W er den såkaldte formfaktor (formmodstand).



Figur 6.2.5 Formfaktoren for nogle forskellige former. Som formel 6.2.8 viser, er luftmodstanden større jo større formfaktoren er. Bevægelsesretningen er mod venstre.
Jakob Strandberg

Det kan betale sig at starte med bare at gætte et k før I beregner den mere grundigt, for at få en ide om hvad det betyder for bevægelsen.

Det som skal ændres er accelerationen i loopet.

```
ddx = -g-0.1*dx*dx          # her har jeg sat k/m=0.1
```

Prøv at kopier hele loopet inklusiv startværdi for `i=0` og plot kommandoen. Så kan I se hvilken forskel det gør at der er luftmodstand.

DATA UD

For at få data ud så I evt. kan sætte det ind i Logger Pro kan I bruger

```
for i in r:
  print(i)
```

Så printer I alt data i `r` som en kolonne som kan markeres og kopieres direkte ind.

FLERE EKSEMPLER

HOOKS LOV

For at simulere hooks lov skal vi lave om på ganske få ting.

Kraften er nu givet ved $F = -k \cdot x$, og vi starter med en startposition forskellige fra nul. I kan også sætte længden af simulationen til noget længere.

Det som skal ændres er

```
T =
x =
dx =
```

I loopet

```
ddx =
```

HOOKS LOV OG KOBLEDE FJEDRE

Hvis vi kalder de to penduler for hhv. x og y så kan vi lave en to af dem. Det bliver først rigtigt sjovt når man får dem til at påvirke hinanden. Vi vil altså lave et system hvor accelerationen af fjedder x bliver påvirket af fjedder y's position.

For fjedder x bliver koden

```
ddx = -k*x + c*y   # hvor c er koblingskonstanten prøv med c=0.2
```

Her er der en del som skal rettes, men prøv.

2D KAST

Samme som I 1D.