LUFTMOTSTAND

python™ & PASCO
koding med mening

I denne øvelsen skal vi undersøke hvordan luft-motstanden avhenger av farten når vi lar muffins-former falle fritt. Vi måler posisjonen som funksjon av tiden og bruker en bevegelsessensor. Normalt ville man vente til posisjon-tid-grafen danner en rett linje, før man fant luftmotstand-koeffisienten k vha. lineær regresjon. Her skal vi derimot vise hvordan vi med Python kan simulere hele hendelsesforløpet og finne rett k, også for vilkårlige objekter med ulike former.

```
# Importering av data med pandas-biblioteket (pd)
csv_data = pd.read_csv(open(filnavn, 'r'),
                  delimiter=";", decimal=",")
t_data = csv_data.iloc[:,1].to_numpy()
s_data = csv_data.iloc[:,2].to_numpy()
t_data = t_data[~isnan(t_data)]
s_data = s_data[~isnan(s_data)]
```

Dette forsøket er fint til å studere teorien for bevegelse med luftmotstand ved å simulere bevegelseslikningene og Newtons 2.lov. Vi setter opp det numeriske forsøket med ingen hastighet, og høyde over sensoren slik vi har rigget det fysiske forsøket. Summen av kreftene er gitt av tyngdekraften $G = mg$ og luftmotstanden $L = kv^2$.

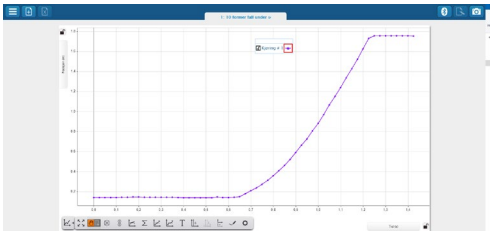
```
# Konstanter til bruk i modellen
m = 0.30*10**(-3)       # Masse, kg
g = 9.81                # Tyngdeakselerasjon, m/s^2
k = 0.0017              # Luftmotstandstallet, kg/m
dt = 0.0001             # Tidssteget som brukes i simuleringen
```

I denne Python-koden er det lurt å innføre funksjoner, hvor mange kan kalle funksjonen a(v) på samme måte som et ordinært funksjonsuttrykk, f(x). Dette forenkler koden betydelig og gjør den mer leselig. Denne funksjonen løser i praksis Newtons' 2.lov for 10 muffins former sammen med en gitt fart, v.

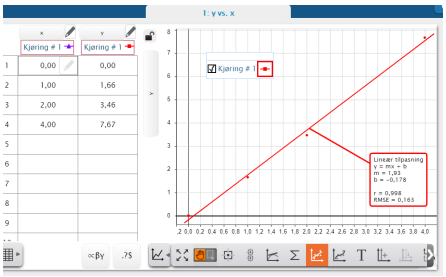
```
# Akselerasjonsfunksjon
def a(v):
    # Utregning av krefter
    G = m*g             # Tyngdekraft, N
    L = k*v**2          # Luftmotstand, N

    # Slippes formen fra under sensor skal sum_F = G - L
    # Slippes formen fra over sensor skal sum_F = L - G
    sum_F = L - G       # Kraftsum, N
    aks = sum_F/m       # Akselerasjon, m/s^2
    return aks
```

Fortsetter neste side…

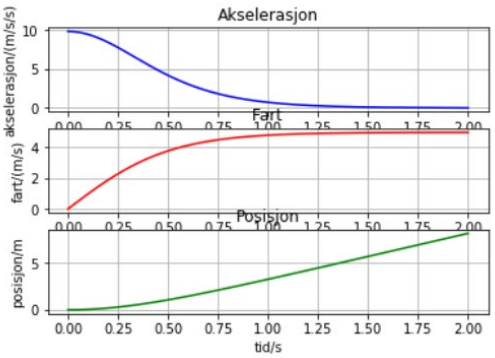
Utstyr
PS-3219 Bevegelsessensor og muffinsformer eller kaffefilter, SPARKvue eller Capstone.



SPARKvue: s-t grafen for 10 muffinsformer i hverandre. Bevegelsessenoren er montert over formene.



Luftmotstanden som funksjon av hastigheten kvadrert i SPARKvue.



Teorien tilsier at akselerasjon, fart og posisjon for muffinsformen vil oppføre seg som i eksemplene over (posisjon måles fra topp til bunn, med positiv retning nedover).

LUFTMOTSTAND

python™ & PASCO
koding med mening

Den mest sentrale delen av koden ser slik ut; Her løser vi bevegelseslikningene for hvert eneste tidssteg

```
# Løkke for simulering av bevegelsen
while t < t_slutt:      # Så lenge muffinsformen faller
    v = v + a(v)*dt     # Regner ut neste fart
    s = s + v*dt        # Regner ut neste posisjon
    t = t + dt          # Regner ut neste tidspunkt

    v_sim.append(v)     # Lagrer den nye farten i liste
    s_sim.append(s)     # Lagrer den nye posisjonen i liste
    t_sim.append(t)     # Lagrer den nye tiden i liste
```

I begynnelsen av koden kan vi gjette oss fram til den rette verdien av luftmostandstallet (koeffisienten) *k*. Vi kan også velge når i datasettet vi slipper formen, som i dette eksempelet er ved tidssteg (datamåling) nr.12. Verdien av k bestemmer hvor bratt den blå kurven til høyre blir, og der vi velger å starte forsøket forskyver datapunktene mot høyre/venstre slik at det passer med startpunktet til den blå grafen.

Vi kan også la koden prøve alle tenkelige verdier av *k* selv, regne ut differansen mellom simulerte verdier og målte datapunkter. Da må koden som regner ut bevegelseslikningene legges inn i som en egen funksjon, «*simulering(s,v,t,a)*», og settes inn i en løkke som velger de ulike verdiene av *k*.

```
# Finne den beste verdien for k
t_slutt = t_data[-1] + dt # Sluttid for simulering
score = 100               # Startscore
for k in k_verdier:       # Verdier for k å iterere over
    s = s_data[0]         # Startposisjon for simulering
    v = 0                 # Startfart for simulering
    t = 0                 # Starttidspunkt for simulering
    t_sim, s_sim = simulering(s,v,t,a)  # Lagrer simulering
    # Kalkulerer differanse
    score_temp = differanse(t_sim, s_sim, t_data, s_data)
    if score_temp < score:  # Hvis vi har bedre score…
        score = score_temp  # Overskriver score
        best_k = k          # Lagrer beste verdi for k
        best_s_sim = s_sim  # Lagrer den beste simuleringen
```

Den store fordelen med denne tilnærmingen, er at den vil finne rett verdi for *k*, uansett form på det objektet du slipper. Objektet før være lett, men utover det kan du prøve deg frem alle typer objekter du finner i klasserommet.

```
Beste verdi for k er: 0.00176
Gjennomsnittelig differanse per punkt:  0.0166
```