Simulering av fallskjermhopp i Python

kodeskolen simula

kodeskolen@simula.no

I dag skal vi lære å bruke programmeringskonsepter som løkker, arrays og plotting til å simulere et fysisk problem, nemlig fallskjermhopp. Vi gjennomgår fysikken og den matematiske tankegangen sammen før dere får bryne dere på programmeringen på egenhånd. Vi hjelper dere selvsagt dersom dere står fast!

Innhold

1 Fallskjermhopping **3**

2 Differensialligninger **4**

2.1 Uløslighet 5

3 Fremgangsmåte **5**

3.1 Terminalhastighet 6

4 Bruke bevegelsesligningene med luftmotstand **7**

4.1 En mer matematisk tankemåte 7

5 Å programmere fallskjermhoppet **8**

5.1 Hastighetsutvikling 8

5.2 Parametre 9

5.3 Framgangsmåte 9

5.4 Utløse fallskjermen 10

6 Å løse for posisjon **11**

7 Regne ut g-krefter **12**

8 Bonus: Strikkhopp **14**

8.1 Strikkraft 15

8.2 Akselerasjonen 16

9 Kode strikkhopp **18**

1 Fallskjermhopping

I dette kompendiet skal vi studere hvordan hastigheten til en person som hopper i fallskjerm endrer seg over tid. Vi kommer til å finne matematiske ligninger basert på vår fysiske forståelse av de kreftene som virker på hopperen. Vi skal diskutere ligningene sammen, og se på hvorfor ikke greier å løse dem med kun penn og papir. Til slutt skal vi derfor se hvordan vi kan løse slike likninger ved hjelp av programmering.

Et fallskjermhopp består hovedsakelig av to deler. Den første delen er det som skjer før fallskjerm er utløst, denne delen kalles gjerne *fritt fall*, i de fleste sportssammenhenger vil man oppleve ca ett minutt fritt fall per hopp. Etter dette utløser hopperen fallskjermen hvilket drastisk reduserer fallhastigheten. Etter å ha utløst fallskjermen, bruker hopperen så vanligvis mellom ett til tre minutter videre ned til bakken avhengig av hvor stor fallskjerm de bruker og hvor nærme bakken de utløste skjermen.

Før vi begynner å se på kreftene som virker på hopperen er det viktig å bestemme seg for en positiv retning. I dette tilfellet er det enklest å bestemme at positiv retning er *nedover*, altså den retningen hopperen faller i.

I begge faser utsettes hopperen bare for to krefter, tyngdekraft og luftmotstand. Så vi har

$$\sum F = F_G - F_D,$$

der F_G er tyngdekraften og F_D er luftmotstand. Dere har sikkert kjennskap til at tyngdekraften er gitt ved $F_G = mg$, der m er den totale massen til hopperen, som vil si personvekten og alt utstyret, og g er tyngdens akselersjonskonstant, som vi setter til $g = 9.81$ m/s^2. En fallskjermhopper vil falle med veldig høy hastighet, og vi bruker derfor en kvadratisk form på luftmotstanden, som vi kan skrive som følger

$$F_D = \frac{1}{2}\rho C A v^2,$$

her er ρ tettheten til luft, C er luftmotstandskoeffisienten som avhenger av formen på objektet som faller og A er frontarealet. For å slippe å skrive så mye kaller vi disse tallene samlet for D, slik at

$$\frac{1}{2}\rho C A = D,$$

$$F_D = Dv^2.$$

ligningen vi skal løse har altså følgende form

$$ma = mg - Dv^2.$$

Siden $a = v'(t)$ kan vi skrive det slik:

$$mv'(t) = mg - Dv^2.$$

Dette er det som matematisk kalles en differensialligning, la oss diskutere hva det betyr.

2 Differensialligninger

Tidligere når vi har sett på ligninger, så er det et matematisk uttrykk, der vi løser for en ukjent. For eksempel kan vi ha ligningen

$$x^2 + 7 = 56.$$

Dette er en ligning fordi vi har en "likhet". Og vi vet at vi kan løse denne for x, ved å flytte over konstanten 7, og så ta roten. Da finner vi at

$$x = \pm 7.$$

Altså har vi løst ligningen, og funnet at x er 7.

En differnsialligning er også en ligning fordi den innholder en "likhet" på samme måte, og den har en ukjent vi løser for. Den store forskjellen er at den ukjente vi løser for, er ikke lenger et enkelt tall, men det er derimot en funksjon.

Newtons 2. lov er et perfekt eksempel på en differensialligning, som vi løser for å finne enten farten, eller posisjonen, til et objekt. Her er både farten, og posisjonen, eksempler på funksjoner, fordi de endrer seg med tid: $v(t)$, $x(t)$. En differensialligning vil innholde den deriverte av funksjonen vi er ute etter, som er mer synlig om vi skriver akselerasjonen som den deriverte av hastigheten i Newtons 2. lov:

$$\sum F = ma = mv'(t)$$

Differensialligningen vi prøver å løse er:

$$mv'(t) = mg - Dv(t)^2.$$

Så, hvordan løser vi en slik ligning? Vel, det finnes utrolig mange forskjellige typer differensialligninger, og det finnes enda flere metoder før å løse dem! De som tar R2, kommer til å lære en del om å løse denne type ligninger, men vi har ikke tid til å dekke så altfor mye av det pensumet her.

2.1 Uløslighet

Differensialligningen vi ønsker å løse er

$$mv'(t) = mg - Dv(t)^2,$$

og vi ønsker å løse den for hastigheten $v(t)$. Men dette får vi ikke til, fordi akselerasjonen avhenger av hastigheten, og motsatt. Dette er grunnen til at vi gjerne ser bortifra luftmotstand når vi løser Newtons 2. lov. Dette går ofte bra fordi resultatet er tilnærmet likt med og uten. Men, i visse tilfeller vil luftmotstand gi et veldig stort forskjell. Fallskjermhopping er et slikt eksempel, uten luftmotstand ville det vært en ganse anderledes opplevelse!

Vi løy faktisk bittelitt i avsnittet over. Det eksisterer differensiallikninger vi kan løse, dere kommer faktisk til å lære om dette i R2. Problemet her er ikke at $v'(t)$ er avhengig av $v(t)$, men at $v'(t)$ er avhengig av $v(t)^2$. Dette kalles for en ikkelineær differensiallikning og dere kommer til å lære om slike likninger om dere tar matematikkfag på universitetet.

Så, hvordan kan vi løse en uløselig ligning ved hjelp av programmering? La oss prøve å ta det steg for steg.

3 Fremgangsmåte

Aller først starter vi med å se på hvordan det hadde gått om vi hadde sett bortifra luftmotstand. Da er $F_D = 0$, slik at:

$$ma = \sum F = mg.$$

Altså er akselerasjonen konstant

$$a = g.$$

I det tilfellet hadde vi kunnet bruke bevegelsesligningene, som hadde gitt oss svarene

$$v(t) = v_0 + at.$$

og

$$x(t) = x_0 + v_0 t + \frac{1}{2}at^2.$$

Altså ser vi at hastigheten vokser konstant for alltid. Dette kan i virkeligheten ikke stemme, da vi vet at en fallskjermhopper vil treffe *terminalhastighet* ganske fort.

3.1 Terminalhastighet

Terminalhastigheten er den raskeste hastigheten et objekt kan falle med, og vi kan enkelt finne den uten å løse differensialligningene vår, vi vet at når luftmotstanden er like stor som tyngdekraften, så vil summen av kreftene på hopperen være 0, og da er akselerasjonen null og falleren faller med en konstant hastighet, nemlig terminalhastigheten.

$$\begin{aligned}\sum F &= ma \\ &= m \cdot 0 \\ &= 0\end{aligned}$$

Vi vet at vi også kan uttrykke $\sum F$ som

$$\begin{aligned}\sum F &= F_G - F_D \\ \sum F &= mg - \frac{1}{2}\rho C A v^2\end{aligned}$$

og når vi setter disse sammen får vi

$$\begin{aligned}0 &= mg - \frac{1}{2}\rho C A v_T^2 \\ mg &= \frac{1}{2}\rho C A v_T^2\end{aligned}$$

Vi har altså

$$mg = \frac{1}{2}\rho C A v_T^2,$$

når vi løser denne for terminalhastigheten har vi

$$v_T = \sqrt{\frac{2mg}{\rho C A}},$$

og når vi setter inn rimelige verdier for et menneske ute fallskjerm ($m = 90$ kg, $C = 1.4$, $\rho = 1$ kg/m^3, $A = 0.7$ m^2, $g = 9.81$ m/s^2) får vi

$$v_T = 42.4 \text{ m/s} = 153 \text{ km/h}.$$

4 Bruke bevegelsesligningene med luftmotstand

Om vi nå legger til luftmotstanden igjen, vet vi at vi ikke kan bruke bevegelsesligningene, fordi vi ikke har konstant akselerasjon. Vi har

$$a(v) = g - \frac{1}{m}Dv^2,$$

og ettersom at hastigheten øker, så vil akselerasjonen minke med tiden. Merk at vi skriver $a(v)$, fordi akselerasjonen er en funksjon av hastigheten. Om vi derimot ser på en veldig liten tidsendring Δt, så vet vi at hastigheten vil endre seg veldig lite, så vi kan bruke bevegelsesligningene et kort *steg* frem i tid ved å si at akselerasjonen er konstant en kort tid:

$$v_1 = v_0 + a(v_0)\Delta t.$$

Nå har vi altså funnet hastigheten til hopperen en kort tid etter han startet. Vi kan nå gå enda litt lenger frem i tid, ved å oppdatere akselerasjonen, og si at den er konstant en ny kort tidsperiode

$$v_2 = v_1 + a(v_1)\Delta t.$$

Trikset her, er å la Δt være veldig liten, slik at akselerasjonen er tilnærmet konstant. Vi må derfor ta veldig mange slike små tidsteg:

$$v_{n+1} = v_n + a(t_n)\Delta t,$$

og på denne måten kan vi løse differensialligningen vår et skritt av gangen til vi har funnet hele løsningen.

4.1 En mer matematisk tankemåte

Differensialligningen vår er

$$a = g - \frac{D}{m}v^2,$$

og vi kan skrive akselerasjonen som den deriverte av hastigheten

$$v'(t) = g - \frac{D}{m}v^2.$$

Definisjonen av den deriverte av v er

$$a = v'(t) = \lim_{\Delta t \to 0} \frac{v(t + \Delta t) - v(t)}{\Delta t}.$$

Vi kan si at vi tilnærmer den deriverte ved istedenfor å la Δt gå helt mot 0, at vi stopper den på et lite tall, vi har da at

$$a = v'(t) \approx \frac{v(t + \Delta t) - v(t)}{\Delta t}.$$

slik at

$$v(t + \Delta t) = v(t) + a\Delta t.$$

Vi ser altså at dersom vi kjenner farten ved tiden t, så kan vi finne den en kort tid senere ved å plusse på $a\Delta t$, der vi kan late som"akselerasjonen er konstant.

5 Å programmere fallskjermhoppet

Det er nå på tide å simulere fallskjermhoppet. Når man skal programmere noe komplisert er det alltid lurt å begynne med en enklere versjon, og vi skal derfor begynne med å simulere den delen av hoppet hvor hopperen er i fritt fall. Når denne delen er klar og i orden legger vi til utløsningen av fallskjermen.

5.1 Hastighetsutvikling

Som dere har sett får vi følgende uttrykk for hastighetsutviklingen:

$$v_{n+1} = v_n + a(v_n)\Delta t,$$

for $n = 0, 1, 2, 3, \ldots$.

Husk at uttrykket vårt for akselleresjonen som en funksjon av hastigheten, $a(v)$ er gitt ved

$$a(v) = g - \frac{1}{2m}\rho C A v^2.$$

For $a(v_n)$ blir dette

$$a(v_n) = g - \frac{1}{2m}\rho C A v_n^2.$$

Slik fortsetter vi til vi har simulert nok tidssteg til å nå en slutt-tid vi har valgt på forhånd. Vi ønsker gjerne at Δt skal være minst mulig, slik at vi får et mer nøyaktig resultat. For eksempel kan vi bruke $\Delta t = 0.01$ s i vår simulering, altså tar vi skritt på ett hundredelssekund frem i tid av gangen. I et vanlig fallskjermhopp

er hopperen ca. 1 minutt i fritt fall, så vi lar sluttiden være $T = 60$ sekunder. Det betyr at vi trenger å ta totalt

$$n = \frac{T}{dt} = \frac{60 \text{ s}}{0.01 \text{ s}} = 6000,$$

tidssteg.

5.2 Parametre

Tallene m, g, ρ, C, A er det som kalles fysiske parametere, og det er størrelser vi velger - vi velger dem utifra hva slags simulering vi ønsker å gjøre, men vi regner dem altså generelt sett som kjent. I vår simulering kan vi bruke følgende parametere:

Parameter	Tallverdi
m	90 kg
g	9.81 m/s^2
ρ	1 kg/m^3
Uten fallskjerm	
C	1.4
A	0.7 m^2
Under fallskjerm	
C_p	1.8
A_p	44 m^2

Det er vanlig å definere alle parametre helt i begynnelsen av programmet.

5.3 Framgangsmåte

Vi begynner med å lage et program hvor hopperen er i fritt fall:

1 Importer numpy og matplotlib, det er alt vi kommer til å trenge.

2 Skriv inn alle parameterene vi trenger, det vil si m, g, ρ, A, C, A_p, C_p, v_0.

2 Lag en funksjon `luftmotstandsstyrke(rho, C, A)` som tar in ρ, C og A og returnerer D.

3 Definer akselerasjonen som en funksjon.
Hint: `def a(v,D,m):`, og husk å returnere noe!

4 Definer $\Delta t = 0.01$ (Hint: kall variablen `dt` i programmet ditt), $T_1 = 60$ og $N_1 = T_1/dt$ (Husk å gjøre N_1 om til et heltall)

5 Opprett to *arrays*, ett for hastigheten v og et for tiden t. Vi vil at de skal være tomme, og ha plass til N_1 elementer, så bruk `zeros` kommandoen. Merk at nå vil `v[n]` i programmet ditt svare til v_n i matematikken.

6 Lag en `for`-løkke som går over $i = 0, 1, 2, \ldots, N_1 - 1$. (Hint, bruk `range`.)

7 Inne i løkka, regn ut $v[n+1]$ fra $v[n]$ ved å bruke formelen vi har funnet. Oppdater også tiden (Hint: `t[n+1] = t[n] + dt`).

8 Plot resultatet for å sjekke at alt har blitt gjort riktig (Hint: `plt.plot(t,v)`).

5.4 Utløse fallskjermen

Vi har nå kommet til tidspunktet der vi skal løse ut fallskjermen i programmet vårt. Det eneste som endrer seg når fallskjerm løses ut, er frontarealet A, og luftmotstandskoeffisienten C (se tabellen med parametre). Ellers er fysikken helt lik. Vi har hitill simulert fallskjerm hopperen i $T = 60$ sekunder, la oss øke denne tiden til 180 sekunder. Men vi lar fortsatt den første løkka bare gå over de første 60 sekundene, deretter endrer vi C og A, og løser for de siste 120 sekundene, altså:

9 Lag en variabel $T_2 = 180$ og en variabel $N_2 = T_2/dt$ og endre størrelsen på v og t arrayene dine slik at de har plass til $N_2 - 1$ elementer.

10 Lag en ny for-løkke som itererer over $n = N_1, N_1 + 1, ..., N_2 - 1$ og regner ut farten og akselerasjonen med parameterne som skal brukes mens fallskjermen er utløst.

11 Plot resultatet for å se om fallskjermhopperen beveger seg saktere etter at fallskjermen løses ut.

6 Å løse for posisjon

Nå har vi et uttrykk for farten til en fallskjermhopper, men vi er ikke fornøyde med det. Vi vil også finne ut hvor langt fallskjermhopperen har falt. Før vi ser på det, er det lurt å huske hvordan vi fikk uttrykket for farten. Vi startet med dette uttrykket for akselerasjonen til fallskjermhopperen, som funksjon av farten hans:

$$a(v) = g - \frac{1}{2}C\rho Av^2.$$

Så brukte vi bevegelseslikningene som gjelder ved konstant akselerasjon,

$$v = v_0 + at,$$

til å ta små tidsstegpå Δt. Dette ga oss denne sammenhengen:

$$\begin{aligned} v_1 &= v_0 + a(v_0)\Delta t \\ v_2 &= v_1 + a(v_1)\Delta t \\ v_3 &= v_2 + a(v_2)\Delta t \\ &\dots \\ v_{i+1} &= v_i + a(v_i)\Delta t. \end{aligned}$$

Når vi vil finne uttrykket hvor langt fallskjermhopperen har falt, kan det være lurt å huske bevegelsesligningen som gjelder for posisjon

$$x = x_0 + v_0 t + \frac{1}{2}at^2.$$

Vi kan nemlig bruke denne til å ta tidsstegpå samme måte som for hastigheten og få dette uttrykket for hvor langt fallskjermhopperen har falt:

$$\begin{aligned} x_1 &= x_0 + v_0\Delta t + \frac{1}{2}a(v_0)\Delta t^2 \\ x_2 &= x_1 + v_1\Delta t + \frac{1}{2}a(v_1)\Delta t^2 \\ x_3 &= x_2 + v_2\Delta t + \frac{1}{2}a(v_2)\Delta t^2 \\ &\dots \\ x_{i+1} &= x_i + v_i\Delta t + \frac{1}{2}a(v_i)\Delta t^2. \end{aligned}$$

7 Regne ut g-krefter

Dere har kanskje hørt om g-krefter, og har kanskje til og med en anelse om hva dette betyr. Her skal vi finne ut hvordan vi kan regne ut g-kreftene til fallskjermhopperen vår, men først: Hva er egentlig g-krefter?

Begrepet g-krefter er faktisk noe misvisende. Det ikke krefter"man oppleverm men en akselerasjon. Når menneskekroppen blir akselerert, føler vi dette som vekt. Tenkt for eksempel når du sitter i en bil som kjører gjennom en sving, og du blir dratttil siden. Disse akselerasjonene føles altså ut som en slags kraft på kroppen, og det er derfor vi snakket om g-krefter. Bokstaven g"i g-kraft står for gravitasjon, og det er fordi vi sammenligner den kraften en person føler med tyngdekraften. Når du står helt i ro, føler du $1g$ fra tyngdekraften.

Ta for eksempel en berg-og-dalbane, der vil man oppleve mange forskjellige g-krefter etterhvert som man akselerer inn og ut av svinger og opp og ned bakker. Når vi har et hurtig dykk nedover på en berg-og-dalbane er man i fritt fall og man blir derfor vektløs, som er $0g$. Etter at man har vært i fritt fall beveger berg-og-dalbanen seg ofte raskt oppover. Da vil man bli trykket ned i setet sitt, og det kjennes ut som om man er tyngre enn vanlig, altså opplever man mer enn $1g$.

En viktig fordel med å snakke om g-krefter fremfor de faktiske fysiske kreftene som virker på et objekt, er at de ikke avhenger av massen. Alle personer vil kjenne de samme g-kreftene i den samme berg-og-dalbanen, uavhengig av hvor tunge de er.

Nå ønsker vi altså å regne ut disse g-kreftene". Dette kan vi gjøre med denne formelen

$$g^{(\text{krefter})} = 1 - a/g,$$

hvor a er akslerasjonen til objektet, g er tyngdeakselerasjonen og positiv retning er nedover mot jorden.

For å regne ut g-kreftene hopperen føler i begge tilfeller trenger vi egentlig bare å legge til et nytt array i løkka, hvor vi regner ut akselerasjonen som virker på hopperen, deler på g og legger til 1. Koden er som følger:

```
gkrefter = zeros(N+1)
...


for i in range(N):
    t[i+1] = t[i] + ...
    v[i+1] = v[i] + ...
    x[i+1] = x[i] + ...
    gkrefter[i] = 1 - a(v[i])/g
```

Nå skal vi regne ut hvor langt fallskjermhopperen har falt, og hvor sterke g-krefter fallskjermhopperen kjenner.

1 Oppdater koden du skrev for å regne ut hastigheten til fallskjermhopperen slik at den også regner ut hvor langt han har falt. (Hint: Lag en ny array, `x`, som du lagrer disse verdiene i)

2 Oppdater koden du skrev for å regne ut g-kreftene fallskjermhopperen kjenner.

3 Plot g-kreftene til fallskjermhopperen, ser du noe rart?

Når du studerer g-kreftene fallskjermhopperen kjenner, så ser du at han opplever nesten $100g$. Det er ikke helt gunstig, siden da vil fallskjermhopperen dø før han treffer bakken! Dette skjer jo ikke i virkeligheten, så da må vi spørre oss selv, hva i fysikken vår er feil?

Det som ikke stemmer er jo at fallskjermen løses ut momentant. Det tar jo noen sekunder fra man trekker i fallskjermsnora til fallskjermen løses ut. Måten vi gjør det er å la luftmotstandskoeffisienten (C) og arealet (A) gå gradvis fra verdiene de har uten fallskjerm til verdiene de har med fallskjerm.

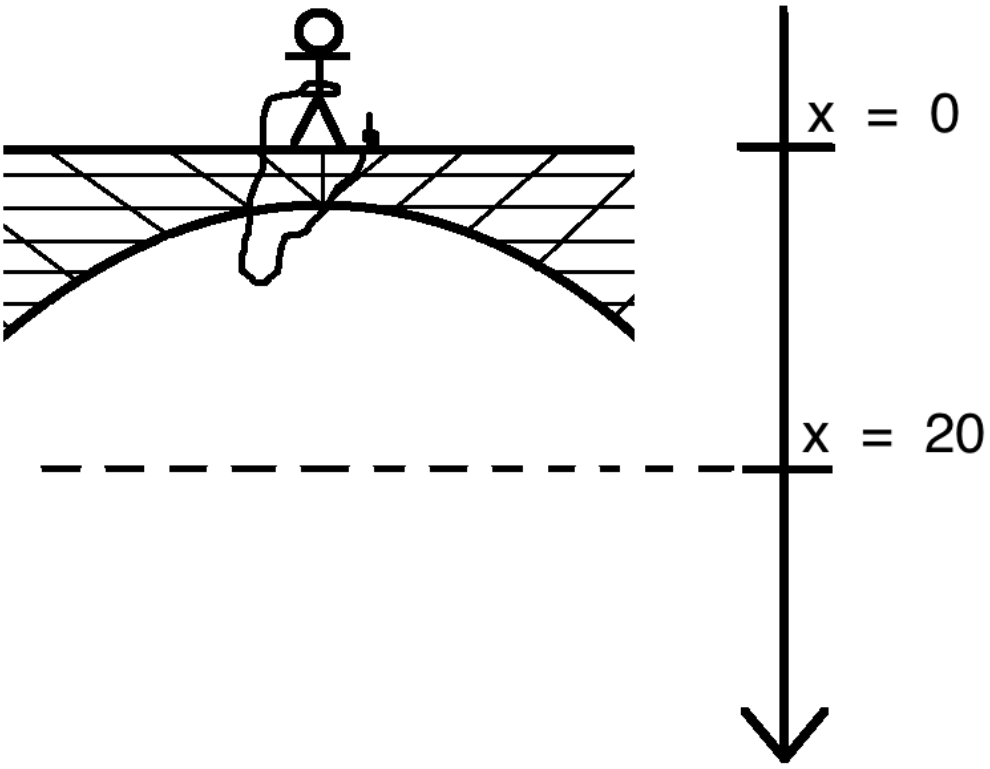
```python
# Simuler de første 60 sekundene uten fallskjerm
...

# Variabler vi trenger for å la fallskjermen løses ut
   gradvis
utløsningstid = 5 # sekunder
utløsnings_N = int((T1 + utløsningstid)/dt)
delta_C = C_p - C
delta_A = A_p - A

# Simulerer de neste sekundene
for i in range(N1, utløsnings_N-1):
    C += delta_C/(utløsningtid/dt)
    A += delt_C/(utløsningstid/dt)

    D = luftmotstandsstyrke(rho, C, A)

    t[i+1] = t[i] + dt
    v[i+1] = v[i] + a(v[i], D, m)*dt
    gforces[i] = 1 - a(v[i], D, m)/g

# Simuler de siste sekundene med fallskjerm
for i in range(utløsnings_N, N2-1):
    ...
```

8 Bonus: Strikkhopp

La oss nå se på en strikkhopper og kreftene som virker på henne. Vi kan tenke på strikken som en slags fjær når den er strukket ut. Men når strikken er trykketss-ammen gir den ingen kraft (i motsetning til en fjær, som virker begge veier"). Når hopperen hopper fra toppen av en bru, er strikken slynget sammen, og kommer derfor ikke til å påvirke hopperen før hun har falt langt nok til at strikken begynner å bli strukket ut. Vi kaller punktet der strikken akkurat ikke er begynnt å bli strukket ut for likevektspunktet.

La oss nå lage et referansesystem som passer bra til problemet vi ser på. Vi plasserer $x_0 = 0$ på toppen av brua som hopperen hopper fra. Vi sier at strikken er 20 meter lang, så likevektspunktet er ved $x = 20$. La oss også si at det går en elv under

brua, og at brua er 60 m over elva.



8.1 Strikkraft

Vi ser nå at dersom posisjonen til hopperen er over likevektspunktet, så vil strikken ikke være strukket ut, og den virker ikke med noen kraft på hopperen. Siden vi har plassert likevektspunktet i $x = 20$, betyr dette at strikkraften, som vi vil kalle S, er 0 hvis $x < 20$.

Hvis $x \geq 20$ derimot, ser vi at strikken er strukket ut, og vil dermed trekke hopperen oppover med en fjærkraft. Denne fjærkraften modellerer vi med det som kalles *Hooke's lov*, som sier at kraften fra en fjær som er strukket ut en lengde x er gitt ved:

$$F = kx,$$

der k er fjærstivheten.

Vi kan altså uttrykke den kraften fra strikken som en funksjon av posisjonen x

som følger:

$$S(x) = \begin{cases} 0 & \text{hvis } x < 20 \\ kx & \text{hvis } x \geq 20 \end{cases}.$$

Måten man skriver dette på i Python er kanskje mer intuitiv enn i matte:

```
if x < 20:
    S = 0
else:
    S = k*x
```

8.2 Akselerasjonen

I tilegg til snorkraften $S(x)$, virker det en tyngdekraft mg, og en luftmotstand Dv, på hopperen. Fra Newtons 2. lov har vi da

$$\begin{aligned} \sum F &= ma \\ \sum F &= F_G - S(x) \pm F_D \\ ma &= F_G - S(x) \pm F_D \\ ma &= mg - S(x) \pm Dv^2 \\ a &= g - \frac{S(x)}{m} \pm \frac{Dv^2}{m} \end{aligned}$$

Som vi nå ser er akselerasjonen en funksjon av både posisjon og hastighet:

$$a(x, v) = g - \frac{S(x)}{m} \pm \frac{Dv^2}{m}$$

Hvorfor står det $\pm$ foran uttrykket for luftmotstanden? Jo, luftmotstanden må jo virke i motsatt retning av hastigheten, så vi må igjen ha et oppdelt uttrykk som vi hadde for kraften fra strikken:

$$a(x, v) = \begin{cases} g - \frac{S(x)}{m} - \frac{Dv^2}{m} & \text{hvis } v > 0 \\ g - \frac{S(x)}{m} + \frac{Dv^2}{m} & \text{hvis } v < 0 \end{cases}$$

I Python ser det slik ut:

Ettersom at akselerasjon nå er en funksjon av både hastighet og posisjon, må vi løse for hastighet og posisjon samtidig, så vi har

$$
\begin{aligned}
v_1 &= v_0 + a(x_0, v_0)\Delta t \\
x_1 &= x_0 + v_0\Delta t + \frac{1}{2}a(x_0, v_0)\Delta t^2 \\
v_2 &= v_1 + a(x_1, v_1)\Delta t \\
x_2 &= x_1 + v_1\Delta t + \frac{1}{2}a(x_1, v_1)\Delta t^2 \\
&\dots \\
v_{n+1} &= v_n + a(x_n, v_n)\Delta t \\
x_{n+1} &= x_n + v_n\Delta t + \frac{1}{2}a(x_n, v_n)\Delta t^2
\end{aligned}
$$

9 Kode strikkhopp

Du skal nå endre programmet du skrev sist uke til å løse for strikkhopping. Her er malen på hva programmet skal inneholde

1. Importer numpy, det er alt vi kommer til å trenge.
2. Skriv inn alle parameterene vi trenger. Bruk $m = 60$, $v_0 = 0$, $x_0 = 0$, $D = 10$. Gjett på en verdi for fjærstivheten k, vi kommer til å justere den etterhvert.
3. Definer snorkraften $S(x)$. Her må du bruke `def` til å definere en funksjon, og en `if`-test inne i funksjonen for å sjekke om $x < 20$ eller $x >= 20$.
4. Definer akselerasjonen som funksjon av både posisjon og hastighet. `def a(x,v):`. Her må du også ha en `if`-test inne i funksjonen for å sjekke om $v < 0$ eller $v > 0$.
5. Definer $\Delta t = 0.01$ (Hint: kall variablen `dt` i programmet ditt), $T = 60$ og $N = T/dt$
6. Opprett fire *arrays*, en for hastigheten v, en for posisjonen x, en for tiden t og en for g-kreftene. Vi vil at de skal være tomme, og ha plass til N elementer, så bruk `zeros` kommandoen.
7. Lag en `for`-løkke som går over $i = 0, 1, 2, \ldots, n-1$. (Hint, bruk `range`.)
8. Inne i løkka, regn ut t_{n+1}, v_{n+1} og x_{n+1}. Fra de følgende formlene

$$\begin{aligned} t_{n+1} &= t_n + \Delta t, \\ v_{n+1} &= v_n + a(x_n, v_n)\Delta t, \\ x_{n+1} &= x_n + v_n \Delta t + \frac{1}{2} a(x_n, v_n)\Delta t^2 \\ g_n^{(\text{krefter})} &= a(v_n)/g \end{aligned}$$

   Vi har altså

```
for n in range(N-1):
    t[n+1] = t[n] + ...
    v[n+1] = v[n] + ...
    x[n+1] = x[n] + ...
    gforces[n] = ...
```

9. Plot resultatet for å sjekke at alt har blitt gjort riktig (Hint: `plot(t,x)`).

10. Pynt på plottet ditt. Sett navn på akser osv.

11. Ved å se på plottet, prøv å juster fjærstivheten k sånn at hopperen akkurat rører vannet. Altså at toppen av kurven akkurat treffer 60.

12. Skriv ut makshastigheten hopperen opplever. Hint: `max(v)`. Hvordan er dette sammenlignet med makshastigheten til fallskjermhopperen?

13. Sjekk at hopperen faktisk overlever g-kreftene den blir utsatt for, og sammenlign med kreftene fallskjermhopperen opplever.