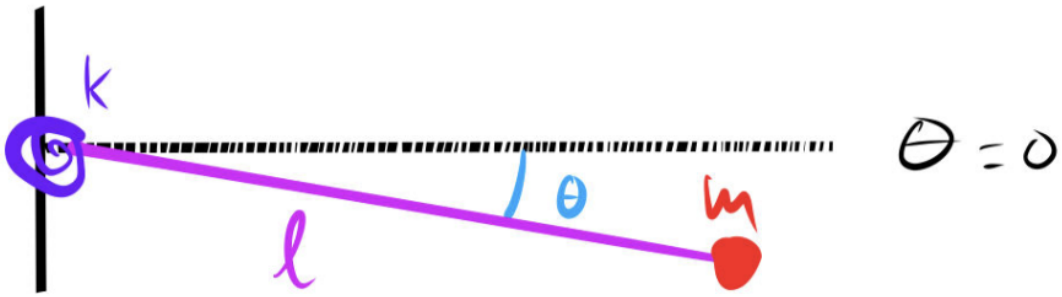
1 Innlevering 3 - Likningsløsning

De fleste likninger som detter ut av praktiske anvendelser er umulige å løse for hånd. Under ser du et bilde av en torsjonsfjær. Dette er en slik fjær du finner i musefeller og i gamle dører og den leverer et moment som er proporsjonalt med utslagsvinkelen θ.

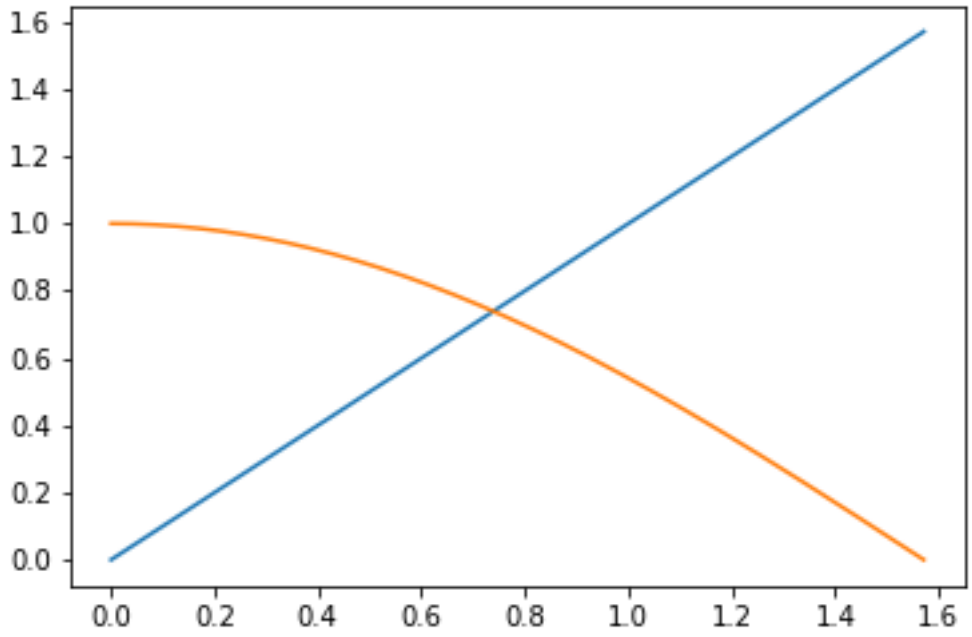


Vi fester en horisontal stang med lengde l i en vegg med en torsjonsfjær med stivhet k og legger en vekt m ytterst på stangen. Gravitasjonskraften fra vekten gir dreiemomentet $ml\cos\theta$, så hvis vi har lyst til å regne ut hvor langt ned vekten drar stangen, må vi finne punktet der momentet fra fjæren balanserer momentet fra vekten:

$$k\theta = ml\cos\theta$$



Vi setter $ml/k = 1$. Her er et plot av funksjonene $f(\theta) = \theta$ og $g(\theta) = \cos\theta$. Disse krysser hverandre, så løsningen til likningen eksisterer, men den er ikke helt god å finne med penn og papir. Løsningen kalles dottietallet og er oppkalt etter en professor i fransk. Professor Dottie oppdaget at hvis hun skrev inn et tilfeldig tall på en kalkulator og trykket gjentatte ganger på cosinusknappen, endte hun alltid opp på omtrent 0.739085.



1. Løs likningen $\theta = \cos\theta$ med vanlig fikspunktiterasjon. Spiller x_0 noen rolle? Hvor mange interasjoner trengs for å nå maskinpresisjon?

På skolen lærte du Newtons metode:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Denne leter etter nullpunkter til f.

2. Løs likningen $\theta = \cos\theta$ med Newtons metode. Spiller x_0 noen rolle? Hvor mange interasjoner trengs for å nå maskinpresisjon?

Som du ser av oppgavene over, oppfører disse to metodene seg helt forskjellig. Dersom vi skal forstå hvorfor, trenger vi Taylors teorem, som sier at dersom f er $n+1$ ganger kontinuerlig deriverbar på et åpent intervall som inneholder x og x, finnes det en s mellom x og a slik at

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \frac{1}{n!}\int_a^x (x-s)^n f^{(n+1)}(s)\, ds$$

Dersom $n = 0$ kalles denne sekantsetningen.

3. Bruk Taylors teorem $n = 0$ til å vise at for fikspunktiterasjonen finnes det, dersom g er deriverbar og alle brouwers teoremers mor og så finnes s slik at

   $$x_{n+1} - r = g'(s)(x_n - r)$$

   der $r = g(r)$ og $x_{n+1} = g(x_n)$.

Dersom man innser at Newtons metode også er en fikspunktiterasjon

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = g(x_n)$$

kan vi analysere den på liknende vis, men vi må sette $n = 1$ i Taylors teorem.

4. Vis at under noen betingelser på fandens oldemor finnes s slik at

$$x_{n+1} - r = \frac{f''(s)}{2f'(x_n)}(r - x_n)^2.$$

Jacobis metode

$$x_{n+1} = D^{-1}\left(b - (L+U)x_n\right)$$

for lineære systemer er forresten også en fikspunktiterasjon. At den konvergerer når A er strengt diagonaldominant er lett å se dersom du kan Banachs fikspunktteorem og vet hva en gershgorinsirkel er.

5. Vis dette og test Jacobis metode på likningssystemene

$$\begin{aligned} 2x_1 + 3x_2 + 4x_3 &= 4 \\ 3x_1 + 4x_2 + 5x_3 &= 5 \\ 4x_1 + 5x_2 + 7x_3 &= 3 \end{aligned}$$

og

$$\begin{aligned} -\frac{1}{\sqrt{2}}x_1 + \frac{1}{2}x_2 &= 0 \\ \frac{1}{\sqrt{2}}x_1 + \frac{\sqrt{3}}{2}x_2 &= 1 \end{aligned}$$

Prøv å spå om det konvergerer før du kjører.

Og her er litt kode for cobwebplot forresten.

[ ]:
```
# pakkene vi trenger
%matplotlib inline
import math # vi trenger tilgang til sinus-funksjonen math.sin()
import numpy as np
import matplotlib.pyplot as plt

# fikser litt på pyplot
newparams = {'figure.figsize': (8.0, 4.0), 'axes.grid': True,
             'lines.markersize': 8, 'lines.linewidth': 2,
             'font.size': 14}
plt.rcParams.update(newparams)

# høyresiden i den logistiske ligninga
def f(x,r,K):
    return x+r*x*(K-x)

#parametre
r=1/2 #vekstfaktor
K=1 #bærekapasitet
u0=1/4 #startverdi
nmax=10 # vi itererer nmax ganger
```

```
n = np.arange(0.0, nmax+1, 1)

# itererer u_{n+1} = f(u_n,r,K) nmax ganger; vi starter på (0,u_0)
u = np.zeros(nmax+1) # vi begynner med en vektor u som har nullere overalt
u[0] = u0
punkter = [(0,u0)]

for i in range(1, nmax+1):
    u[i] = f(u[i-1], r, K)
    ui = u[i]
    punkter.append((i,ui))

plt.plot(n, u, 'bo')
print(punkter)

# cobweb-plottet

x = np.arange(0, 1., 0.01) # lager mesh for grafer på en annen måte enn␣
 ↪tidligere

# vi plotter y = f(x,r,K) og y = x, hvor x representerer u_n
plt.plot(x, f(x,r,K), 'b') # plotter y = f(x,r,K) i blått
plt.plot(x, x, 'r')        # plotter y = x i rødt

# visualiser coweb i grønt
plt.plot([u[0],u[0]], [0,u[1]],'g') # første vertikale linje mellom punktene␣
 ↪(u[0],0) og (u[0],u[1])
for i in range(1, nmax):
    plt.plot([u[i],u[i]],[u[i],u[i+1]],'g') # lager vertikal linje mellom␣
 ↪punktene (u[i],u[i]) og (u[i],u[i+1])

for i in range(0, nmax):
    plt.plot([u[i],u[i+1]],[u[i+1],u[i+1]],'g') # lager horisontal linje mellom␣
 ↪punktene (u[i],u[i+1]) og (u[i+1],u[i+1])

plt.show()
```