FYS2130 Prosjektoppgave

Kandidatnummer: 15069

20. mai 2017

Innhold

Oppgave 1

I denne oppgaven bruker jeg de tre ligningene oppgitt i oppgaveteksten.
Den første ligningen uttrykker kraften på et massepunkt via de to nærliggende fjærene. Fjærkraften bestemmes entydig av en fjærkonstant og strekningen av fjæren. Dette er standard Hooke's lov $F = -k\Delta y$. Når bevegelsen til massepunktene er begrenset til å gå langs y-aksen er vi kun interessert i kraften som går i denne retningen. Strekningen bestemmes da av den relative forskjellen i posisjon langs y-aksen til massepunktene. Figur O.1 i oppgaveteksten illustrerer hvordan man kan komme frem til konkrete uttrykk for kraften fra fjæren til venstre og høyre.

$$F_{i,v} = -k_{i-1}\,\mathrm{d}y_{i,v} = -k_{i-1}(y_i - y_{i-1})$$
$$F_{i,h} = -k_i\,\mathrm{d}y_{i,h} = -k_i(y_i - y_{i+1})$$

Ligning O.1 slår sammen kreftene for å få et uttrykk for totalkraften som virker på et vilkårlig massepunkt i.

$$F_i = F_{i,v} + F_{i,h} = -(k_{i-1} + k_i)y_i + k_{i-1}y_{i-1} + k_i y_{i+1} \tag{O.1}$$

Den andre ligningen tilnærmer den dobbelt tidsderiverte av posisjonen via forholdstallet mellom differanseverdier av nabomassepunkter og et lite tidsintervall Δt. Dette er svært likt den vanlige definisjonen av den deriverte som en differenskvotient, bare brukt to ganger. Som vanlig gjelder denne diskretiseringen kun for små Δt (mer og mer nøyaktig i grensen når Δt går mot 0)

$$\ddot{y}_i = \frac{\mathrm{d}^2 y_i}{\mathrm{d}t^2} \approx \frac{\dfrac{y_i^+ - y_i^0}{\Delta t} - \dfrac{y_i^0 - y_i^-}{\Delta t}}{\Delta t} = \frac{y_i^+ - 2y_i^0 + y_i^-}{(\Delta t)^2} \tag{O.2}$$

Ligning tre er en omskrevet Newton's andre lov. Indeksen viser hvilket massepunkt det er snakk om. Summen av krefter på denne må være produktet av massen til punktet og akselerasjonen den opplever.

$$F_i = m_i \ddot{y}_i \tag{O.3}$$

Generell løsning av oppgaven

Når det skal løses for y_i^+ med de spesifiserte variablene kan man ta utgangspunkt i ligning O.3. Herfra kan man erstatte F_i på venstre side via ligning O.1 og $\ddot{y}_i$ på høyre side via ligning O.2. Deretter er det ren algebra for å løse ligningen for y_i^+.

$$F_i = m_i \ddot{y}_i$$
$$-(k_{i-1} + k_i)y_i + k_{i-1}y_{i-1} + k_i y_{i+1} = m_i\left(\frac{y_i^+ - 2y_i^0 + y_i^-}{(\Delta t)^2}\right) \tag{1}$$
$$y_i^+ = \frac{(\Delta t)^2}{m_i}[-(k_{i-1} + k_i)y_i + k_{i-1}y_{i-1} + k_i y_{i+1}] + 2y_i^0 - y_i^-$$

Her utføres enkel multiplikasjon og divisjon før addisjon og subtraksjon i det siste steget for å få y_i^+ alene (Klammeparantes brukes for å lettere kunne skjelne de separate leddene).
Overgangen fra den kontinuerlige til den diskrete versjonen av $\ddot{y}_i$ i ligning O.2 gjelder som sagt kun for små Δt, så dette kravet gjelder også for sluttuttrykket.

Som poengtert i oppgaven vil det ikke være mulig å løse dette uttrykket for endepunktene y_0^+ og y_{N-1}^+ i modellen vår fordi det på den ene siden av hvert av disse punktene værken finnes en fjær eller nytt massepunkt. For det første punktet $i = 0$ eksisterer ikke $k_{i-1} = k_{-1}$ eller $y_{i-1} = y_{-1}$. For det siste punktet $i = N - 1$ finnes ikke $k_i = k_{N-1}$ (indeksen på den siste fjæren er $i = N - 2$) eller $y_{i+1} = y_N$ (Indeksen på det siste massepunktet er $i = N - 1$).
Dette problemet kan løses på mange forskjellige måter. I oppgaven trekkes det frem to metoder.

Åpne randbetingelser

Ved åpne ender eller åpne randbetingelser (open boundary conditions) bruker man kun verdiene fra den ene siden og ignorerer den andre. I vårt tilfelle betyr dette at $F_0 = F_{0,h}$ for massepunktet helt til venstre på bølgen og $F_{N-1} = F_{N-1,v}$ på høyre side. Med dette kan vi løse for endepunktene eksplisitt.

$$y_0^+ = \frac{(\Delta t)^2}{m_0}[-k_0(y_0 - y_1)] + 2y_0^0 - y_0^-$$

$$y_{N-1}^+ = \frac{(\Delta t)^2}{m_{N-1}}[-k_{N-2}(y_{N-1} - y_{N-2})] + 2y_{N-1}^0 - y_{N-1}^-$$

Reflekterende randbetingelser

Systemer med eflekterende randbetingelser eller reflekterende ender har endepunkter som er låst fast og aldri beveger seg. Én måte å implementere dette er å sette massen til endepunktene svært høyt slik at de ikke beveger seg selv om store krefter virker på dem. Reflekterende ender har fått dette navnet fordi alle bølger som treffer dem reflekteres tilbake nesten uten å tape energi (tilnærmet uendelig impedans).

$$y_0^j = y_{N-1}^j = 0$$

Her indikerer indeksen j at dette gjelder for alle tidssteg.

Oppgave 2

En konstant massetetthet $\mu = m/\Delta x$ og konstant fjærstivhet $\kappa = k\Delta x$ introduseres. Hvis man antar at Δx også er konstant, det vil si at den er lik mellom alle massepunktene (dette står man fritt til å gjøre og er det mest naturlige valget hvis modellen brukes på et virkelig makroskopisk tilfelle, for eksempel et svingende tau), må nødvendigvis også k og m være konstante. Hvis man setter inn disse konstante verdiene for fjærkraft og masse ($k_{i-1} = k_i = k$ og $m_i = m$) inn i ligning 1) kan man utlede en diskretisert versjon av bølgelingningen.

$$
\begin{aligned}
-(k+k)y_i + ky_{i-1} + ky_{i+1} &= m\left(\frac{y_i^+ - 2y_i^0 + y_i^-}{(\Delta t)^2}\right) \\
k(y_{i-1} - 2y_i + y_{i+1}) &= m\left(\frac{y_i^+ - 2y_i^0 + y_i^-}{(\Delta t)^2}\right) \\
\frac{\kappa}{\Delta x}(y_{i-1} - 2y_i + y_{i+1}) &= \mu\Delta x\left(\frac{y_i^+ - 2y_i^0 + y_i^-}{(\Delta t)^2}\right) \\
\frac{\kappa}{\mu}\left(\frac{y_{i-1} - 2y_i + y_{i+1}}{(\Delta x)^2}\right) &= \frac{y_i^+ - 2y_i^0 + y_i^-}{(\Delta t)^2}
\end{aligned}
$$

En diskretisert versjon av den dobbelt posisjonsderiverte av y, med samme form som den vi hadde for den tidsderiverte i ligning O.2, dukker opp på venstre side av ligningen. I grensen når Δx og Δt går mot null når vi den kontinuerlige versjonen som gjenkjennes som bølgeligningen.

$$
\frac{\partial^2 y}{\partial t^2} = \kappa/\mu \frac{\partial^2 y}{\partial x^2}
$$

Omdøper man den konstante faktoren $\kappa/\mu = v_p^2$ ser man at ligningen samsvarer med bølgeligningen for en bølge med fasehastighet

$$
v_p = \sqrt{\frac{\kappa}{\mu}} = \Delta x\sqrt{\frac{k}{m}} \tag{2}
$$

Oppgave 3

For at et program skal være presist nok til å modellere en bølge som beveger seg med en gitt fasehastighet v_p må den numeriske hastigheten til programmet $\Delta x/\Delta t$ være større enn denne fasehastigheten. Ved å sette inn uttrykket for v_p fra forrige oppgave er det mulig å relatere tidsstegene Δt i programmet til massen og fjærkonstanten i masspunktene og fjærene i strengen.

$$\begin{aligned} v_p &< \Delta x/\Delta t \\ \Delta t &< v_p^{-1}\Delta x \\ \Delta t &< \Delta x\sqrt{\mu/\kappa} \\ \Delta t &< \Delta x\sqrt{m/\Delta x^2 k} \\ \Delta t &< \sqrt{m/k} \end{aligned} \tag{3}$$

Fordi Δx forsvinner er denne variabelen uavhengig kravet om oppløsningen og det er kun Δt vi må sørge for er liten nok. Årsaken tydeliggjøres ved å se nærmere på modellen av strengen. Vi ønsker å sørge for at den faktiske avstanden Δx_B amplitudene til bølgen beveger seg med i løpet av ett tidssteg er lenger enn avstanden mellom hvert punkt Δx vi bruker i programmet.

$$\Delta x < \Delta x_B = v_p \Delta t = \Delta t \Delta x\sqrt{k/m} \implies \Delta t\sqrt{k/m} > 1$$

Hvilket er det samme resultatet som i den opprinnelige tilnærmingen.

Oppgave 4

I denne oppgaven blir vi bedt om å implementere programmet som det hittil har blitt utredet for. Utslaget i hvert punkt til bølgen ved starttidspunktet t_i velges

$$y_i^0 = \sin\left(7\pi \frac{i}{N-1}\right), \quad 0 \leq i \leq N-1$$

Fordi argumentet $\frac{i}{N-1}$ alltid vil gå fra null til én er dette en sinusbølge som går fra 0 til 7π. Når 2π indikerer én bølgelengde ser man at det her er snakk om tre og én halv bølgelengde. Her ser vi at endepunktene oppfyller kravet om reflekterende ender når utslaget er null. Det er nå også mulig å se at bølgen oppfyller kravet til en stående bølge med reflekterende ender

$$n\lambda = 2L$$

Et heltallsmultiplum av bølgelengden må være dobbelt så lang som avstanden mellom endepunktene. I vårt tilfelle ser man at $n = 7$. Dette er det også mulig å utlede ved å ta utgangspunkt i definisjonen for en stående bølge. Videre vil jeg anta at vi starter ved tidspunktet $t_i = 0$[1]

$$\begin{aligned} y_i^0 &= A\cos(\omega t_i)\sin(kx) \\ \sin\left(7\pi \frac{i}{N-1}\right) &= A\sin(kx) \\ 7\pi \frac{i}{N-1} &= kx \\ 7\pi \frac{x}{L} &= \frac{2\pi}{\lambda}x \\ 7\lambda &= 2L \end{aligned}$$

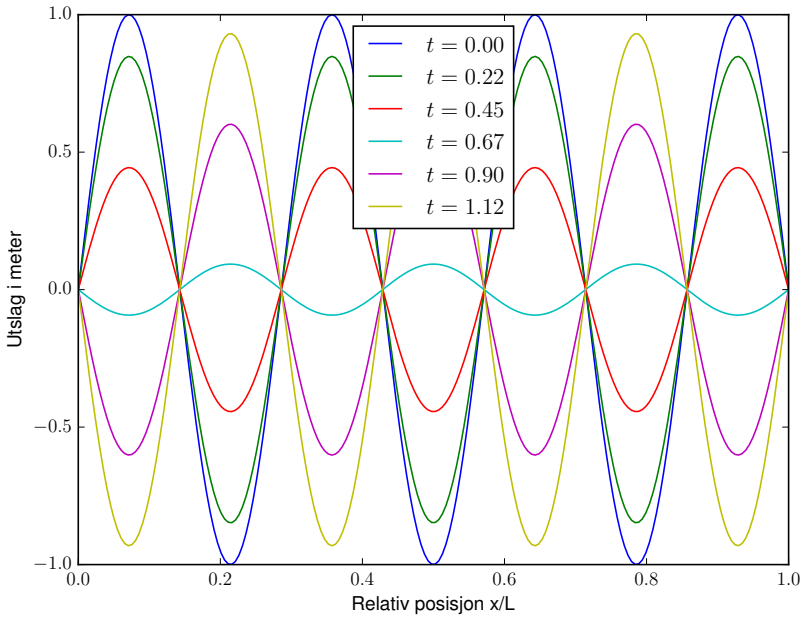
Her er det igjen en sammenheng mellom den kontinuerlige og diskrete situasjonen som dukker opp ved å erstatte forholdet mellom indeks og totalt antall indekser med forholdet mellom posisjonsvariabelen x og total lengde mellom reflekterende punkter. Den velkjente sammenhengen mellom bølgetall og bølgelengden brukes også. Ved å derivere funksjonen for den stående bølgen

[1] Det er fortsatt den samme bølgebevegelsen som blir beskrevet. Å sette $t_i = 0$ betyr bare at vi antar at den stående bølgen starter med maksimalt utslag. Det finnes ikke noe absolutt nullpunkt/referansepunkt for tid og antagelsen er analog med å innføre et faseledd. Fordi $\cos(0) = 1$ blir $A = 1$. Merk at jeg her har valgt å bruke A som den faktiske amplituden til den stående bølgen og ikke $2A$ som ofte brukes under utledningen for stående bølger om amplituden til de to opprinnelige bølgene som bygger opp den stående bølgen.

mhp. tid kan man se hvordan tidsutviklingen til bølgen er.

$$\frac{\partial}{\partial t} A\cos(\omega t)\sin(kx)$$
$$A\sin(kx)\left(\frac{\mathrm{d}u}{\mathrm{d}t}\right)\left(\frac{\mathrm{d}\cos(u)}{\mathrm{d}u}\right)$$
$$-A\sin(kx)\left(\frac{\mathrm{d}\omega t}{\mathrm{d}t}\right)\sin(\omega t)$$
$$-\omega A\sin(kx)\sin(\omega t)$$

Allerede her ser man at ved å velge $t_i = 0$ forsvinner uttrykket og bølgen forandrer seg ikke med tiden. Dette er det vi ville forvente når en stående bølge akkuratt har nådd sitt maksimum og alle punkter er i ferd med å snu og gå mot nullutslag. Dette betyr at



I programmet har jeg valgt tidssteget $\Delta t = \frac{1}{10}\sqrt{m/k}$ ettersom dette oppfyller kravet om hastighetsoppløsning.

Oppgave 5

Det er mulig å finne vinkelfrekvensen til bølgen i oppgave 4 gjennom følgende kjente relasjon

$$\omega = \mathbf{k}v_p = \frac{2\pi}{\lambda}\Delta x\sqrt{\frac{k}{m}} = \frac{7\pi\Delta x}{L}\sqrt{\frac{k}{m}} = \frac{7\pi}{(N-1)}\sqrt{\frac{k}{m}}$$

Her har jeg brukt fet $\mathbf{k}$ for å skille mellom bølgetallet og fjærkonstanten. [2] Dette uttrykket kan brukes videre for å finne svingefrekvensen

$$f = \frac{\omega}{2\pi} = \frac{7}{2(N-1)}\sqrt{\frac{k}{m}} = \frac{7}{2(200-1)}\sqrt{\frac{10\,\mathrm{N/m}}{0.02\,\mathrm{kg}}} \approx 0.3932783\,\mathrm{Hz}$$

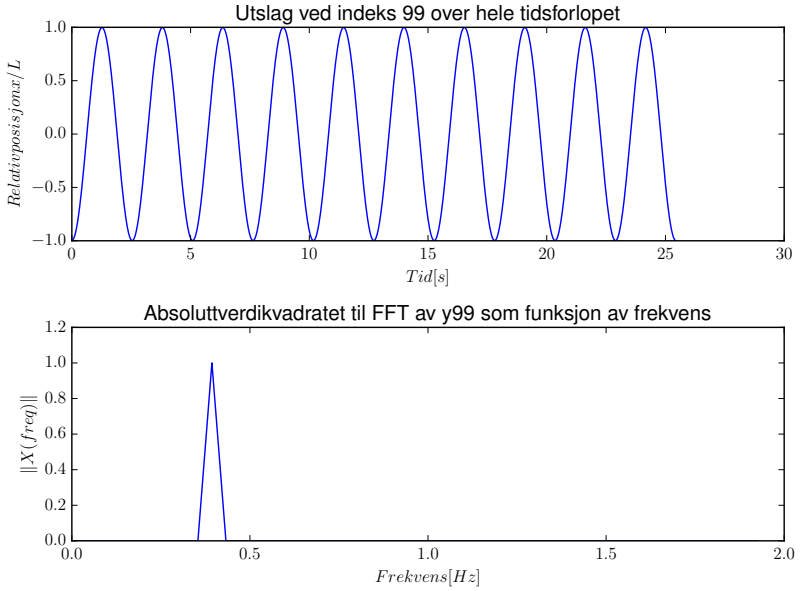
Den inverse relasjonen mellom frekvens og periode gjør det lett å finne de etterspurte ti periodene

$$10T = 10/f = \frac{20(N-1)}{7}\sqrt{\frac{m}{k}}$$

Anntall tidssteg i tallverdi blir da

$$N_t = 1 + \frac{10T}{\Delta t} = \frac{20(N-1)}{7\Delta t}\sqrt{\frac{m}{k}} = \frac{20(200-1)}{7\Delta t}\sqrt{\frac{0.02\,\mathrm{kg}}{10\,\mathrm{N/m}}} \approx 25.43\Delta t^{-1}$$

Fordi antall tidssteg bør være et heltall må denne verdien omgjøres i programmet. I figuren under er utslaget i punkt 100 plottet for hele tidsforløpet, samt fouriertransformasjonen.



[2] andre steder forstås hvilken k det er snakk om ut ifra kontekst

Man ser at punktet starter med -1 i utslag, et bunnpunkt, og svinger ti ganger om likevektspunktet hvor det har null utslag før det ender i bunnpunktet igjen. Plottet under viser fouriertransformasjonen av punktet. Vi går da fra tidsbildet over til frekvensbildet. Fordi bølgen er såpass enkel, med prinsipielt én frekvens, er det plottet over langt færre frekvenser en normalt. Dette er for å få med den lille toppen som ligger ved den lave frekvensen rundt 0.39. Ved å la programmet vurdere tidsforløpet mellom hvert bunnpunkt (se vedlegg) kommer det fram til en frekvens på $f = 0.3929327\,\mathrm{Hz}$. Dette med en relativt god presisjon når det kun var én av åtte frekvenser (vurdert mellom åtte bunnpunkter) som returnerte en annen verdi. Dette er ganske nærme verdien for frekvensen som ble funnet analytisk. Forholdet mellom den numeriske og analytiske frekvensen er

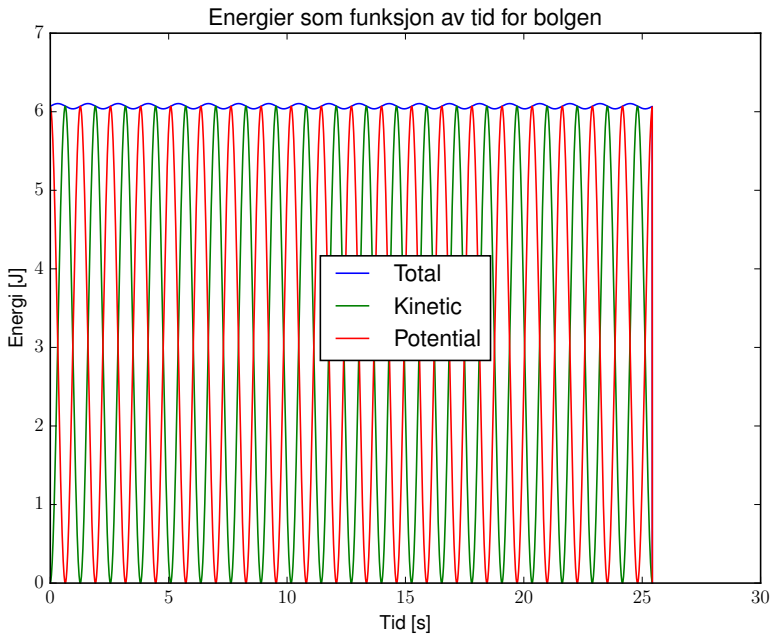
$$\frac{f_n}{f_a} = 0.999121$$

Oppgave 6

For å finne ut hvordan energien til systemet utvikler seg over tid brukes en løkke som iterer over hvert tidssteg (med unntak av det aller siste. Under hvert tidssteg regnes den kinetiske og potensielle energien ut, E_k og E_p. Den totale mekaniske energien er summen av disse. For hver indeks i kalkuleres den kinetiske energien til massepunktet, $E_{k,i}$, og den potensielle energien til fjæren, $E_{p,i}$.

$$E_{p,i} = \frac{1}{2}k\Delta x^2 = \frac{1}{2}k(y_{i+1} - y_i)^2$$
$$E_{k,i} = \frac{1}{2}mv^2 = \frac{1}{2}m\left(\frac{y_i^+ - y_i^0}{\Delta t}\right)^2$$
$$E_i = E_{p,i} + E_{k,i}$$

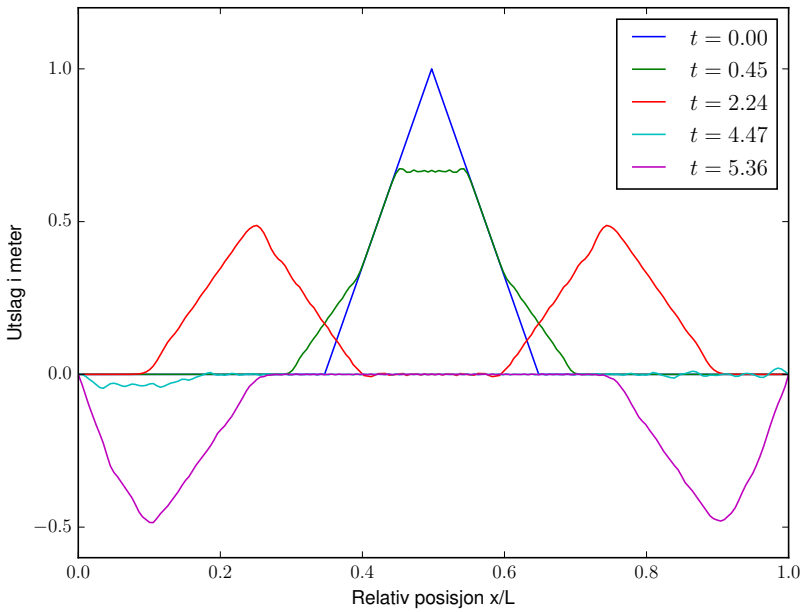
Det totale antall fjærer er én færre enn massepunkter. $E_{p,N-1}$ finnes altså ikke, mens $E_{k,N-1}$ gjør det. Til gjengjeld kan ikke hastigheten til massepunktene regnes ut i det siste tidssteget med denne metoden.



Plottet viser at den kinetiske og potensielle energien er motsatt proporsjonale. Når den stående bølgen er helt flat (null utslag over hele linja) er all energi kinetisk. Når bølgen har maksimale utslag er all energi potensiell. Den blå linjen øverst viser at totalenergien ikke er fullstendig bevart. Dette kan være grunnet numeriske beregningsfeil.

Oppgave 7

I denne oppgaven byttes initialbetingelsene for y_i^- og y_i^0 til en trekant (definert i ligning O.6). Plottet kommer her lett ut når det eneste vi trenger å forandre er disse initialbetingelsene.



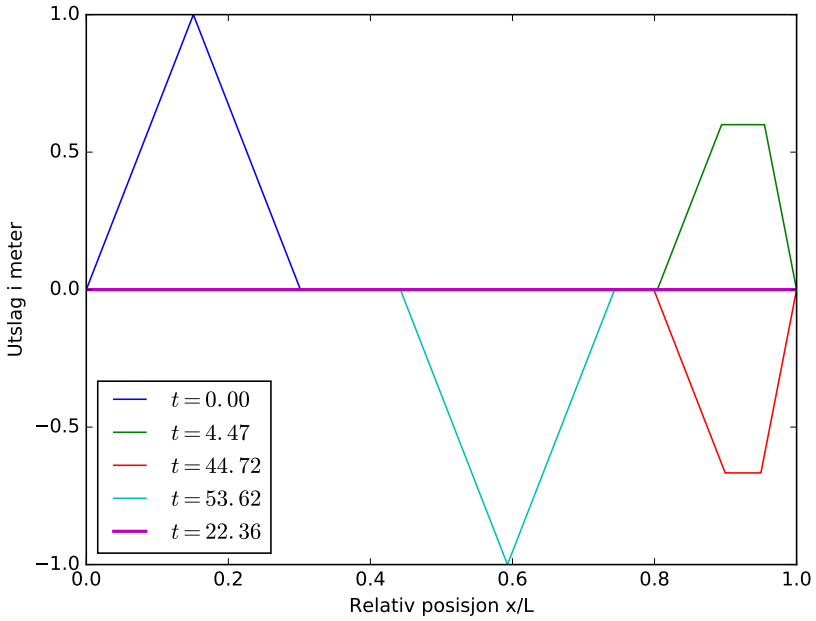
Her er trekanten plottet fra $t_i = 0$ til $t_f = 5.37$. Fordi kreftene i massepunktene i skråningen til trekanten utligner hverandre (hvert punkt blir trukket like mye opp som ned) er det kun i de diskontinuerlige punktene ved toppen og sidekantene som beveger seg til å begynne med. Kreftene forplanter seg med tiden og vi ser etter hvert at to trekanter, som beveger seg i motsatt retning av hverandre, dukker opp. Den opprinnelige trekanten kan tenkes som en superposisjon av disse to trekantbølgene. Hadde vi sent de to mindre trekantene mot hverandre med samme fart (bare motsatt retning) ville den opprinnelige trekanten dukket opp. Vi ser igjen at når bølgene når endepunktene snur de slik at amplituden bytter fortegn før de blir reflektert og traverserer i motsatt retning.

Oppgave 8

For å få hele trekanten til å bevege seg mot høyre er det nok å sette initialbetingelsen y_i^- ett tidssteg til venstre for den orginale trekanten.

$$y_i^- = y_{i+1}^0$$

Fordi trekanten ender opp med å forstyrre randen i $y_0^- = 1/30$ er denne nødt til å holdes på plass eksplisitt ved utregningen av y_i^+ ved det første tidssteget. Dette gjøres ved å se når iterasjonsvariabelen for tidsstegene er $j = 0$ og da sette $y_0 = 0$. For alle andre tidssteg brukes den vanlige reflekterende randbetingelsen. Det at vi er nødt til å gjøre dette eksplisitt kan tolkes i at informasjon om trekantens venstre endepunkt/hjørne ble mistet da den ble forflyttet til venstre.



Oppgave 9

Den enkleste måten å regne impedansforholdet er å bruke at, for en streng modellert slik vi har gjort, er impedansen på et linjestykke med en gitt masse og fjærkonstant gitt ved

$$Z = \sqrt{km}$$

Ikke ulikt det vi fant for hastigheten (lignende variabler og vekstfaktor). Dette bør ikke være overaskende når bølgehastigheten er relatert til impedansen. Bølger beveger seg raskere i et medium med lav impedans enn et med høyere. Når massen til den ene delen av strengen er tre ganger så stor som den andre og fjærkonstanten er den samme ser vi at

$$\frac{Z_1}{Z_0} = \sqrt{\frac{3km}{km}} = \sqrt{3}$$

Hvor Z_1 er impedans på den siste halvdelen og Z_0 er på den første. For å finne hvordan impedansforholdet relateres til amplitude kan man bruke uttrykkene for transmisjonskoeffisient og refleksjonskoeffisient.

$$T = \frac{A_T}{A_I} = \frac{2Z_0}{Z_0 + Z_1} \tag{4}$$

$$R = \frac{A_R}{A_I} = \frac{Z_0 - Z_1}{Z_0 + Z_1} \tag{5}$$

$$\frac{T}{R} = \frac{A_T}{A_R} = \frac{2Z_0}{Z_0 - Z_1}$$

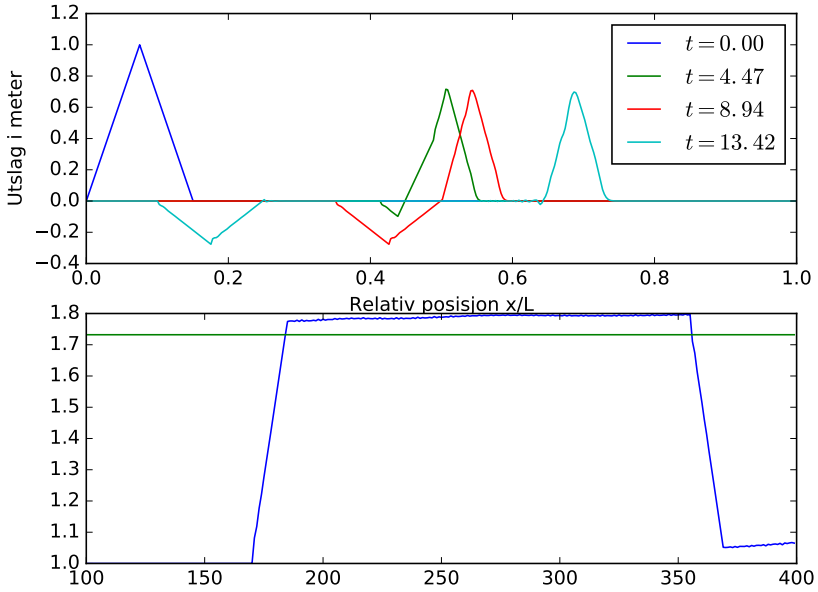
$$Z_0 - Z_1 = 2Z_0 \frac{A_R}{A_T}$$

$$\frac{Z_0}{Z_0} - \frac{Z_1}{Z_0} = s2\frac{Z_0}{Z_0}\frac{A_R}{A_T}$$

$$1 - \frac{Z_1}{Z_0} = 2\frac{A_R}{A_T}$$

$$\frac{Z_1}{Z_0} = 1 - 2\frac{A_R}{A_T}$$

Under er plottet for trekantbølgen for fire forskjellige tider og et plot over impedansforholdet for en del tidssteg.



Man ser tydelig at den reflekterte bølgen beveger seg raskere enn den transmitterte. Den grønne linjen i plottet under viser at forholdet er litt større enn det burde (rundt 1.77)

Notat

I denne oppgaven har jeg samarbeidet med kandidatnummer 15104 som hjalp meg med tolkningen av initialbetingelse av bølgen i oppgave 4 og uttrykk for impedans i forhold til amplitude i oppgave 9.

Kode til oppgave 4

```
from numpy import *
from matplotlib.pyplot import *
figure(figsize = (8,6))
rc('text', usetex = True)

# --== INITIAL CONDITIONS AND CONSTANTS ==--

# Initialization of x-value array
Nx = 200 # total number of x-values
xi = 0 # initial x-value
xf = 1 # final x-value
dx = float64((xf-xi) / (Nx-1)) # distance between x-values
x = linspace(xi,xf,Nx) # array of all x-values

# Initialization of time values array
Nt = 1200 # number of timesteps
ti = 0 # initial time
dt = float64(0.1*sqrt(0.02/10)) # time between timesteps
tf = ti + Nt * dt # final time
t = linspace(ti,tf,Nt) # Array of all times

# Arrays containing point masses and spring constants
m = 0.02*ones(Nx)
m[0] = 1E6*m[0] # greatly increase the mass of endpoints
m[-1] = 1E6*m[-1]
k = 10*ones(Nx-1)

# Constants deduced from analytical solution of the wave
kw = 7*pi/(xf-xi) # Wavenumber
omg = kw*dx*sqrt(k[0]/m[0]) # Angular frequency of the wave
A = 1/cos(omg*ti) # Amplitude of the wave

# Initial waves y0 and y-
y = zeros((len(t),len(x))) # Nested array of waves. y[i,:] is the wave at t[i]
y[0] = sin(7*pi*arange(Nx)/(Nx-1)) # Initial amplitudes at time t0
yPrev = y[0] + dt * (-A*omg*sin(kw*x)*sin(omg*ti))
# adjusting starting boundaries from numerical error
y[0,0], y[0,-1], yPrev[0], yPrev[-1] = [0,0,0,0]

# A function that returns amplitude at next timestep for a given point
def yNext(dt,m,k,y,yPrev):
    return 2*y[1] - yPrev + ( dt**2/m * ((-k[0]-k[1])*y[1] + k[0]*y[0] + k[1]*y[2]) )
```

```
# --== CALCULATION ITERATIONS ==--
for j in range(0,Nt-1):
    # handling boundary conditions
    y[j+1,0] = yNext(t[j+1]-t[j], m[0], [0,k[0]], [0,y[j,0],y[j,1]], yPrev[0] )
    y[j+1,-1] = yNext(t[j+1]-t[j], m[-1], [k[-1],0], [y[j,-1],y[j,0],0], yPrev[-1] )
    # iterating over all other x to calculate y+
    for i in range(1,Nx-1):
        y[j+1,i] = yNext(t[j+1]-t[j], m[i], k[i-1:i+1], y[j,i-1:i+2], yPrev[i] )
    yPrev = y[j]

# --== PLOTTING ==--
for i in [0,50,100,150, 200, 250]:
    plot(x,y[i],label = '$t=%.2f$' % (t[i]))
xlabel('Relativ␣posisjon␣x/L')
ylabel('Utslag␣i␣meter')
legend(loc = 'best')
savefig('4.pdf')
```

Kode til oppgave 5

```
from numpy import *
from matplotlib.pyplot import *
#figure(figsize = (5,4))
rc('text', usetex = True)

# --== INITIAL CONDITIONS AND CONSTANTS ==--

# Initialization of x-value array
Nx = 200 # total number of x-values
xi = 0 # initial x-value
xf = 1 # final x-value
dx = float64((xf-xi) / (Nx-1)) # distance between x-values
x = linspace(xi,xf,Nx) # array of all x-values

# Arrays containing point masses and spring constants
m = 0.02*ones(Nx)
m[0] = 1E6*m[0]
m[-1] = 1E6*m[-1]
k = 10*ones(Nx-1)

# Initialization of time values array
dt = float64(0.1*sqrt(0.02/10))
ti = 0
tf = sqrt(m[1]/k[1])*20*(Nx-1)/7
Nt = int(1 + (tf-ti)/dt)
t = linspace(ti,tf,Nt) # Array of all times

# Constants deduced from analytical solution of the wave
kw = 7*pi/(xf-xi) # Wavenumber
omg = kw*dx*sqrt(k[0]/m[0]) # Angular frequency of the wave
A = 1/cos(omg*ti) # Amplitude of the wave

# Initial waves y0 and y-
y = zeros((len(t),len(x))) # Nested array of waves. y[i,:] is the wave at t[i]
y[0] = sin(7*pi*arange(Nx)/(Nx-1)) # Initial amplitudes at time t0
yPrev = y[0] + dt * (-A*omg*sin(kw*x)*sin(omg*ti))
y[0,0], y[0,-1], yPrev[0], yPrev[-1] = [0,0,0,0]

# A function that returns amplitude at next timestep for a given point
def yNext(dt,m,k,y,yPrev):
    return 2*y[1] - yPrev + ( dt**2/m * ((-k[0]-k[1])*y[1] + k[0]*y[0] + k[1]*y[2]) )

# --== CALCULATION ITERATIONS ==--
```

```
for j in range(0,Nt-1):
    # handling boundary conditions
    y[j+1,0] = yNext(t[j+1]-t[j], m[0], [0,k[0]], [0,y[j,0],y[j,1]], yPrev[0] )
    y[j+1,-1] = yNext(t[j+1]-t[j], m[-1], [k[-1],0], [y[j,-1],y[j,0],0], yPrev[-1] )
    # iterating over all other x to calculate y+
    for i in range(1,Nx-1):
        y[j+1,i] = yNext(t[j+1]-t[j], m[i], k[i-1:i+1], y[j,i-1:i+2], yPrev[i] )
    yPrev = y[j]

y99 = y[:,99] # picks out magnitude at index 99 for all times

# --== Finding nine extremums ==--
ext = zeros(9)
count = 0
for i in range(len(y99)-2):
    if( y99[i+1] < min([y99[i],y99[i+2]]) ):
        ext[count] = t[i]
        count = count + 1
print(1/(ext[1:]-ext[:-1]))
""" Returns:
[ 0.3929327  0.3929327  0.3929327  0.39362448
0.3929327  0.3929327   0.3929327  0.3929327 ]"""

# --== Fourier transform and plot==--
X = fft.rfft(y99)
X = X/sum(X)
freq = fft.rfftfreq(Nt, d = dt)

subplot(2,1,1)
plot(t,y99)
title('Utslag␣ved␣indeks␣99␣over␣hele␣tidsforlopet')
ylabel('$Relativ␣posisjon␣x/L$')
xlabel('$Tid␣[s]$')

subplot(2,1,2)
title('Absoluttverdikvadratet␣til␣FFT␣av␣y99␣som␣funksjon␣av␣frekvens')
plot(freq[:50],(abs(X)**2)[:50])
xlabel('$Frekvens␣[Hz]$')
ylabel('\\|X(freq)\\|')

tight_layout()
savefig('5.pdf')
```

Kode til oppgave 6

```
from numpy import *
from matplotlib.pyplot import *
#figure(figsize = (5,4))
rc('text', usetex = True)

# --== INITIAL CONDITIONS AND CONSTANTS ==--

# Initialization of x-value array
Nx = 200 # total number of x-values
xi = 0 # initial x-value
xf = 1 # final x-value
dx = float64((xf-xi) / (Nx-1)) # distance between x-values
x = linspace(xi,xf,Nx) # array of all x-values

# Arrays containing point masses and spring constants
m = 0.02*ones(Nx)
m[0] = 1E6*m[0]
m[-1] = 1E6*m[-1]
k = 10*ones(Nx-1)

# Initialization of time values array
dt = float64(0.1*sqrt(0.02/10))
ti = 0
tf = sqrt(m[1]/k[1])*20*(Nx-1)/7
Nt = int(1 + (tf-ti)/dt)
t = linspace(ti,tf,Nt) # Array of all times

# Constants deduced from analytical solution of the wave
kw = 7*pi/(xf-xi) # Wavenumber
omg = kw*dx*sqrt(k[0]/m[0]) # Angular frequency of the wave
A = 1/cos(omg*ti) # Amplitude of the wave

# Initial waves y0 and y-
y = zeros((len(t),len(x))) # Nested array of waves. y[i,:] is the wave at t[i]
y[0] = sin(7*pi*arange(Nx)/(Nx-1)) # Initial amplitudes at time t0
yPrev = y[0] + dt * (-A*omg*sin(kw*x)*sin(omg*ti))
y[0,0], y[0,-1], yPrev[0], yPrev[-1] = [0,0,0,0]

# A function that returns amplitude at next timestep for a given point
def yNext(dt,m,k,y,yPrev):
    return 2*y[1] - yPrev + ( dt**2/m * ((-k[0]-k[1])*y[1] + k[0]*y[0] + k[1]*y[2]) )

# --== CALCULATION ITERATIONS ==--
```

```
for j in range(0,Nt-1):
    # handling boundary conditions
    y[j+1,0] = yNext(t[j+1]-t[j], m[0], [0,k[0]], [0,y[j,0],y[j,1]], yPrev[0] )
    y[j+1,-1] = yNext(t[j+1]-t[j], m[-1], [k[-1],0], [y[j,-1],y[j,0],0], yPrev[-1] )
    # iterating over all other x to calculate y+
    for i in range(1,Nx-1):
        y[j+1,i] = yNext(t[j+1]-t[j], m[i], k[i-1:i+1], y[j,i-1:i+2], yPrev[i] )
    yPrev = y[j]

E = zeros((len(t),3))
for j in range(Nt-1):
    E[j,0] = sum( 0.5*k[1]*(y[j,1:] - y[j,:-1])**2 ) # potential energy
    E[j,1] = sum(0.5*m[1]*((y[j+1,::]-y[j,::])/dt)**2) # kinetic energy
    E[j,2] = E[j,0] + E[j,1]

# -=== PLOTTING ==--
plot(t,E[::,2],label = 'Total')
plot(t,E[::,1],label = 'Kinetic')
plot(t,E[::,0],label = 'Potential')
title('Energier␣som␣funksjon␣av␣tid␣for␣bolgen')
xlabel('Tid␣[s]')
ylabel('Energi␣[J]')
legend(loc = 'center')
savefig('6.pdf')
```

Kode til oppgave 7

```
from numpy import *
from matplotlib.pyplot import *
rc('text', usetex = True)

# --== INITIAL CONDITIONS AND CONSTANTS ==--

# Initialization of x-value array
Nx = 200 # total number of x-values
xi = 0 # initial x-value
xf = 1 # final x-value
dx = float64((xf-xi) / (Nx-1)) # distance between x-values
x = linspace(xi,xf,Nx) # array of all x-values

# Initialization of time values array
Nt = 1200 # number of timesteps
ti = 0 # initial time
dt = float64(0.1*sqrt(0.02/10)) # time between timesteps
tf = ti + (Nt-1) * dt # final time
t = linspace(ti,tf,Nt) # Array of all times

# Arrays containing point masses and spring constants
m = 0.02*ones(Nx)
m[0] = 1E6*m[0] # greatly increase the mass of endpoints
m[-1] = 1E6*m[-1]
k = 10*ones(Nx-1)

y = zeros((len(t),len(x)))
y[0,70:100] = (arange(70,100)-69)/30
y[0,100:129] = (129-arange(100,129))/30
yPrev = y[0]
plot(x,yPrev)

# A function that returns amplitude at next timestep for a given point
def yNext(dt,m,k,y,yPrev):
    return 2*y[1] - yPrev + ( dt**2/m * ((-k[0]-k[1])*y[1] + k[0]*y[0] + k[1]*y[2]) )

# --== CALCULATION ITERATIONS ==--
for j in range(0,Nt-1):
    # handling boundary conditions
    y[j+1,0] = yNext(t[j+1]-t[j], m[0], [0,k[0]], [0,y[j,0],y[j,1]], yPrev[0] )
    y[j+1,-1] = yNext(t[j+1]-t[j], m[-1], [k[-1],0], [y[j,-1],y[j,0],0], yPrev[-1] )
    # iterating over all other x to calculate y+
    for i in range(1,Nx-1):
```

```
        y[j+1,i] = yNext(t[j+1]-t[j], m[i], k[i-1:i+1], y[j,i-1:i+2], yPrev[i] )
    yPrev = y[j]

# --== PLOTTING ==--
plot(x,y[0], label = '$t=%.2f$' % (t[0]))
plot(x,y[100], label = '$t=%.2f$' % (t[100]))
plot(x,y[500], label = '$t=%.2f$' % (t[500]))
plot(x,y[1000], label = '$t=%.2f$' % (t[1000]))
plot(x,y[-1], label = '$t=%.2f$' % (t[-1]))
legend(loc = 'best')
xlabel('Relativ␣posisjon␣x/L')
ylabel('Utslag␣i␣meter')
savefig('7.pdf')
```

Kode til oppgave 8

```
from numpy import *
from matplotlib.pyplot import *
#rc('text', usetex = True)

# --== INITIAL CONDITIONS AND CONSTANTS ==--

# Initialization of x-value array
Nx = 200 # total number of x-values
xi = 0 # initial x-value
xf = 1 # final x-value
dx = float64((xf-xi) / (Nx-1)) # distance between x-values
x = linspace(xi,xf,Nx) # array of all x-values

# Initialization of time values array
Nt = 1200 # number of timesteps
ti = 0 # initial time
dt = float64(1*sqrt(0.02/10)) # time between timesteps
tf = ti + (Nt-1) * dt # final time
t = linspace(ti,tf,Nt) # Array of all times

# Arrays containing point masses and spring constants
m = 0.02*ones(Nx)
m[0] = 1E9*m[0] # greatly increase the mass of endpoints
m[-1] = 1E6*m[-1]
k = 10*ones(Nx-1)

y = zeros((len(t),len(x))) # nested array where element 'i' holds amplitudes for time t=i
y[0,:31] = arange(31)/30
y[0,31:60] = (60-arange(31,60))/30
yPrev = zeros(len(y[0]))
yPrev[:-1] = y[0,1:]

# A function that returns amplitude at next timestep for a given point
def yNext(dt,m,k,y,yPrev):
    return 2*y[1] - yPrev + ( dt**2/m * ((-k[0]-k[1])*y[1] + k[0]*y[0] + k[1]*y[2]) )

# --== CALCULATION ITERATIONS ==--
for j in range(0,Nt-1):
    if(j == 0):
        y[j+1,0] = 0
    else:
        y[j+1,0] = yNext(t[j+1]-t[j], m[0], [0,k[0]], [0,y[j,0],y[j,1]], yPrev[0] )
    y[j+1,-1] = yNext(t[j+1]-t[j], m[-1], [k[-1],0], [y[j,-1],y[j,0],0], yPrev[-1] )
```

```
    # iterating over all other x to calculate y+
    for i in range(1,Nx-1):
        y[j+1,i] = yNext(t[j+1]-t[j], m[i], k[i-1:i+1], y[j,i-1:i+2], yPrev[i] )
    yPrev = y[j]

# -=== PLOTTING ==--
plot(x,y[0], label = '$t=%.2f$' % (t[0]))
plot(x,y[160], label = '$t=%.2f$' % (t[100]))
plot(x,y[179], label = '$t=%.2f$' % (t[1000]))
plot(x,y[250], label = '$t=%.2f$' % (t[-1]))
plot(x,y[169], label = '$t=%.2f$' % (t[500]), lw = 2)
legend(loc = 'best')
xlabel('Relativ␣posisjon␣x/L')
ylabel('Utslag␣i␣meter')
savefig('8.pdf')
```

Kode til oppgave 9

```
from numpy import *
from matplotlib.pyplot import *
#rc('text', usetex = True)

# --== INITIAL CONDITIONS AND CONSTANTS ==--

# Initialization of x-value array
Nx = 400 # total number of x-values
xi = 0 # initial x-value
xf = 1 # final x-value
dx = float64((xf-xi) / (Nx-1)) # distance between x-values
x = linspace(xi,xf,Nx) # array of all x-values

# Initialization of time values array
Nt = 1200 # number of timesteps
ti = 0 # initial time
dt = float64(1*sqrt(0.02/10)) # time between timesteps
tf = ti + (Nt-1) * dt # final time
t = linspace(ti,tf,Nt) # Array of all times

# Arrays containing point masses and spring constants
m = 0.02*ones(Nx)
m[200:] = 3*0.02 # increases the mass of points for the last half by a factor 3
m[0] = 1E9*m[0] # greatly increase the mass of endpoints
m[-1] = 1E6*m[-1]
k = 10*ones(Nx-1)

y = zeros((len(t),len(x)))
y[0,:31] = arange(31)/30
y[0,31:60] = (60-arange(31,60))/30
yPrev = zeros(len(y[0]))
yPrev[:-1] = y[0,1:]

# A function that returns amplitude at next timestep for a given point
def yNext(dt,m,k,y,yPrev):
    return 2*y[1] - yPrev + ( dt**2/m * ((-k[0]-k[1])*y[1] + k[0]*y[0] + k[1]*y[2]) )

# --== CALCULATION ITERATIONS ==--
for j in range(0,Nt-1):
    if(j == 0):
        y[j+1,0] = 0
    else:
        y[j+1,0] = yNext(t[j+1]-t[j], m[0], [0,k[0]], [0,y[j,0],y[j,1]], yPrev[0] )
```

```
    y[j+1,-1] = yNext(t[j+1]-t[j], m[-1], [k[-1],0], [y[j,-1],y[j,0],0], yPrev[-1] )
    # iterating over all other x to calculate y+
    for i in range(1,Nx-1):
        y[j+1,i] = yNext(t[j+1]-t[j], m[i], k[i-1:i+1], y[j,i-1:i+2], yPrev[i] )
    yPrev = y[j]

# --== PLOTTING ==--
subplot(211)
plot(x,y[0], label = '$t=%.2f$' % (t[0]))
plot(x,y[175], label = '$t=%.2f$' % (t[100]))
plot(x,y[200], label = '$t=%.2f$' % (t[200]))
plot(x,y[300], label = '$t=%.2f$' % (t[300]))
legend(loc = 'best')
xlabel('Relativ␣posisjon␣x/L')
ylabel('Utslag␣i␣meter')

subplot(212)
y_rel = (y).min(axis = 1) / (y).max(axis = 1)
plot(arange(100,400),1- 2*( y_rel[100:400] ))
plot(arange(100,400),sqrt(3)*ones(300))
savefig('9.pdf')
```