Prosjektoppgave FYS2130

Kandidatnummer 15091

9. mai 2017

Innhold

1 Oppgaver

1.1 Oppgave 1

Fra oppgaveteksten har vi

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

$$\ddot{y}_i = \frac{d^2 y_i}{dt^2} \quad \simeq \frac{y_i^+ - 2y_i^0 + y_i^-}{(\Delta t)^2} \tag{O.2}$$

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

Setter vi (O.1) og (O.2) inn i (O.3) finner vi at

$$\begin{aligned} F_i &= m_i\left(\frac{y_i^+ - 2y_i^0 + y_i^-}{(\Delta t)^2}\right) \\ F_i \frac{(\Delta t)^2}{m_i} &= y_i^+ - 2y_i^0 + y_i^- \\ y_i^+ &= \frac{F_i (\Delta t)^2}{m_i} + 2y_i^0 - y_i^- \end{aligned} \tag{0.5}$$

$$y_i^+ = (-(k_{i-1} + k_i)y_i + k_{i-1}y_{i-1} + k_i y_{i+1})\frac{(\Delta t)^2}{m_i} + 2y_i^0 - y_i^- \tag{1}$$

Som er et uttrykk for y_i^+. Dette uttrykket er derimot avhengig av $k_{i-1}y_{i-1}$ og $k_i y_{i+1}$ og vil dermed se litt annerledes ut for endepunktene. Fordi (O.1) er den totale fjærkraften fra høyre + venstre, kan vi lett finne den totale fjærkraften dersom massepunktet ikke er koblet til en fjær på en av disse sidene. $F_i = F_{i,h} + 0$ for den første punktmassen (helt til venstre) og $F_i = F_{i,v} + 0$ for den siste punktmassen (helt til høyre). Vi får dermed følgende uttrykk for endepunktene y_0^+ og y_{N-1}^+

$$y_0^+ = -k_i(y_i - y_{i+1})\frac{(\Delta t)^2}{m_i} + 2y_i^0 - y_i^- \tag{1.1}$$

$$y_{N-1}^+ = -k_i(y_i - y_{i+1})\frac{(\Delta t)^2}{m_i} + 2y_i^0 - y_i^- \tag{1.2}$$

Som tilsvarer uttrykk (1), med kun fjærkraft fra den siden som har en fjær koblet til seg.

1.2 Oppgave 2

Konstant massetetthet $\mu = \frac{m}{\Delta x}$ betyr her at hver av våre punktmasser m_i blir lik $\mu\Delta x$, mens fjærkonstanten k_i til fjærene mellom punktmassene våre kan omskrives til $\kappa = k\Delta x \to k = \frac{\kappa}{\Delta x}$. Setter vi dette inn i uttrykket (1) vi fant i oppgave 1 får vi følgende.

$$\begin{aligned} y_i^+ &= \tfrac{\kappa}{\Delta x}(-2y_i + y_{i-1} + y_{i+1})\tfrac{(\Delta t)^2}{\mu\Delta x} + 2y_i^0 - y_i^- \\ \tfrac{y_i^+ - 2y_i^0 + y_i^-}{(\Delta t)^2} &= \tfrac{\kappa}{\Delta x}(-2y_i + y_{i-1} + y_{i+1})\tfrac{1}{\mu\Delta x} \end{aligned} \tag{2}$$

Fra (O.2) blir dette

$$\begin{aligned} \tfrac{d^2y_i}{dt^2} &= \tfrac{\kappa}{\mu(\Delta x)^2}(y_{i+1} - 2y_i + y_{i-1}) \\ \tfrac{d^2y_i}{dt^2} &= \tfrac{\kappa}{\mu}(\tfrac{y_{i+1} - 2y_i + y_{i-1}}{(\Delta x)^2}) \\ \tfrac{d^2y_i}{dt^2} &= \tfrac{\kappa}{\mu}\tfrac{d^2y_i}{dx^2} \end{aligned} \tag{3}$$

Her har vi et uttrykk som ligner veldig på bølgeligningen vi skulle finne. dersom $\frac{\kappa}{\mu} = v_B^2$ har vi fått uttrykket vi skulle utlede (O.4).

$$\tfrac{d^2y_i}{dt^2} = v_B^2\tfrac{d^2y_i}{dx^2} \tag{O.4}$$

Men hva er v_B^2? Utbredelseshastigheten i x-retning $\frac{\Delta x}{\Delta t}$. Vi ser på (O.3) og (O.1). og finner følgende uttrykk for $\frac{\Delta x}{\Delta t}$.

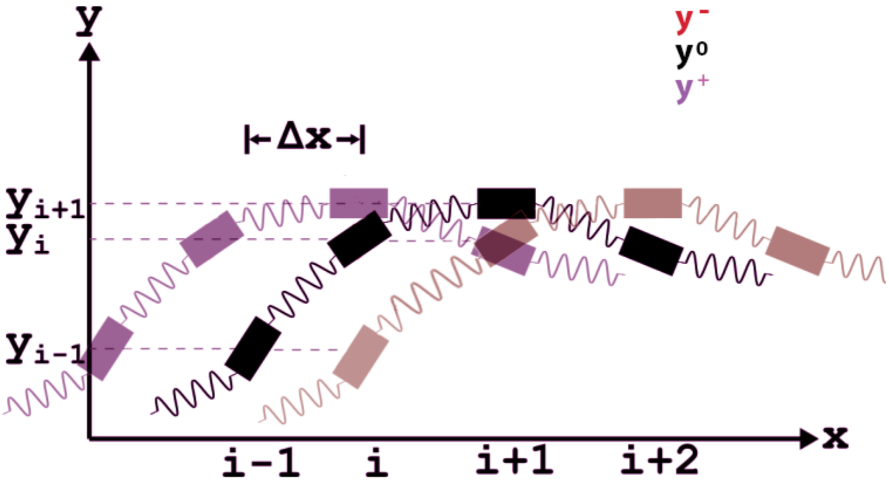
$$\begin{aligned} F = m_i(\tfrac{y_i^+ - 2y_i^0 + y_i^-}{(\Delta t)^2}) &= -(k_{i-1} + k_i)y_i + k_{i-1}y_{i-1} + k_iy_{i+1} \\ \tfrac{m}{(\Delta t)^2}(y_i^+ - 2y_i^0 + y_i^-) &= k(y_{i-1} - 2y_i + y_{i+1}) \\ \tfrac{\mu(\Delta x)^2}{\kappa(\Delta t)^2} &= \tfrac{y_{i-1} - 2y_i + y_{i+1}}{y_i^+ - 2y_i^0 + y_i^-} \\ (\tfrac{\Delta x}{\Delta t})^2 &= \tfrac{\kappa}{\mu}\tfrac{(y_{i-1} - y_i) + (y_{i+1} - y_i)}{(y_i^+ - y_i^0) + (y_i^- - y_i^0)} \end{aligned} \tag{4}$$

Dette vil si at når endringen i y-retning til et massepunkt i tid Δy_t $(y_i^0 \to y_i^+)$ er lik posisjonsendringen Δy_i fra et massepunkt y_i til det neste massepunktet y_{i+1}, får vi

$$
\begin{aligned}
(\tfrac{\Delta x}{\Delta t})^2 = v_B^2 &= \tfrac{\kappa}{\mu}(\tfrac{\Delta y_i + (\Delta y)_{i-1}}{\Delta y_t + (\Delta y)_{t-1}}) \\
v_B^2 &= \tfrac{\kappa}{\mu}
\end{aligned}
\tag{5}
$$

Dersom $\lim\limits_{\Delta y_t \to \Delta y_i}$ vil dette være tilfelle og vi får $v_B^2 = \frac{\kappa}{\mu}$. (hvis $\Delta y_t = \Delta y_i$)[1]

Altså vil utbredelseshastigheten være $v_B^2 = \frac{\kappa}{\mu}$ når punktmassenes y-posisjon $y_i^0 = y_{i+1}^- = y_{i-1}^+$.



Figur 1: Visuell forklaring på punktmassenes bevegelse over tid med gitt v_B^2 (retning venstre) og Δt som jeg har prøvd å uttrykke over.

[1]En annen måte å si det på er $\frac{\Delta y}{\Delta t} = \frac{\Delta y}{\Delta x} \to v_b = v_s$ hvor v_s er svingehastigheten

1.3 Oppgave 3

$$
\begin{aligned}
\frac{\Delta x}{\Delta t} &\geq v_B &= \sqrt{\frac{\kappa}{\mu}} &= \sqrt{\frac{k(\Delta x)^2}{m}} = \sqrt{\frac{k}{m}}\Delta x \\
\frac{\Delta x}{\Delta x} &= 1 &\geq \sqrt{\frac{k}{m}}\Delta t \\
\Delta t &\leq \sqrt{\frac{m}{k}} &= \frac{1}{\omega} = \frac{T}{2\pi}
\end{aligned}
\tag{6}
$$

Altså vi må bruke tidssteg som er mindre enn perioden $\frac{T}{2\pi}$. Numerisk kan vi sørge for at verdien Δt alltid er lav nok ved å sette den lik $c\sqrt{\frac{m}{k}}$ hvor $0 < c \leq 1$.

Fordi vi opererer med Δx-avhengige verdier vil ikke en endring i Δx påvirke Oppløsningen. Vi opererer med punktpartikler uten utstrekning i x-retning (en $\Delta x =$ en punktmasse).

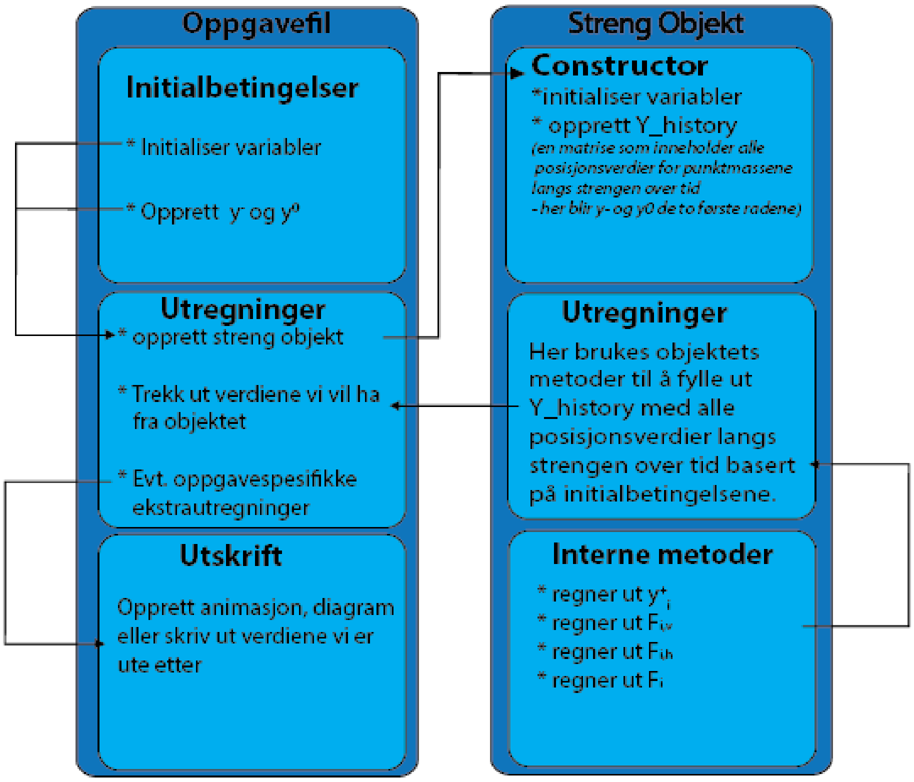
Fra (5) og (6) får vi

$$
\begin{aligned}
\Delta t &\leq \frac{1}{v_B} &= \frac{\Delta t}{\Delta x} \\
\Delta x &\leq \frac{\Delta t}{\Delta t} &= 1
\end{aligned}
\tag{7}
$$

Med punktmasser uten utstrekning i x-retning $\rightarrow \Delta x = 1 \leq 1$. Δt er uavhengig av Δx.

1.4 Programstruktur

I Oppgaveteksten ble det oppgitt et forslag til hvordan programmet skulle se ut (Figur O.2). I denne besvarelsen ble programmet strukturert noe annerledes, men operasjonene vi går gjennom er nokså like.[2]



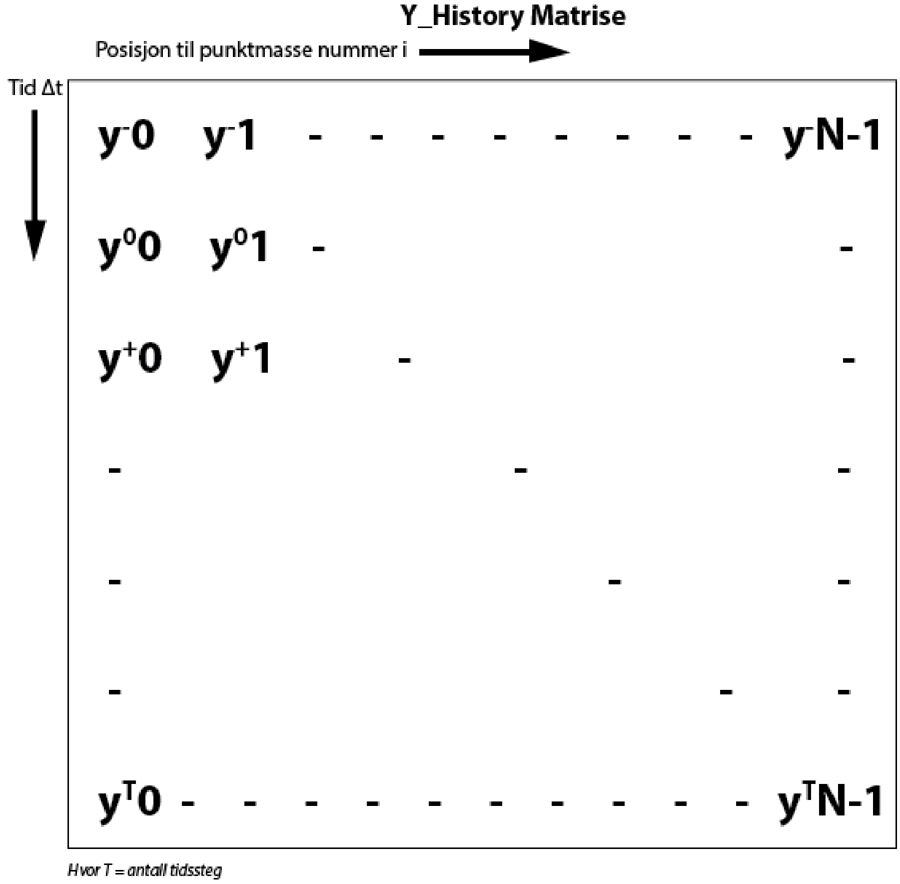
Figur 2: Flow Chart

Alle oppgaveløsningene finner man i en egen fil som først oppretter variablene og initialbetingelsene. De sender dermed dette til et objekt Streng (Vedlegg 1). Grunnen til dette er fordi mye av det programmene skal gjøre er identisk i alle oppgavene. Tanken var dermed at mye av det kunne generaliseres i et objekt.

Streng lagrer først initialbetingelsene lokalt, for så å opprette en $Tidssteg \times N$ matrise y_history. I denne matrisen lagres alle posisjonsveridene i alle

[2]Selv om koden til programmene er lagt til under vedlegg 1, kan man også finne alle programfilene her: http://folk.uio.no/andhel/FYS2130/

tidsstegene som programmet regner ut. Initialbetingelsene y_i^- og y_i^0 lagres dermed som de første 2 radene i y_history.



Figur 3: y_history matrisen i Streng objektet. $T = n\Delta t$ hvor n = antall tidssteg (minus 1 fordi vi begynner på 0).

Fra oppgavefilen bestemmer man dermed hvilken type utregning man er ute etter når Streng objektet kalles med gitte verdier (vil man regne ut y-verdier i en gitt mengde tidssteg N, regne ut nye verdier frem til en viss posisjon har beveget seg en viss periode, en streng med eller uten bunde ender osv.).

Etter posisjonsverdiene er regnet ut og lagret i y_history er resten av programmet skrevet i oppgavefilen. Denne delen av programmene består av sortering, utskrift og animasjon av informasjonen vi har kommet frem til.[3]

[3]Det krever litt mer minne å lagrer alle verdiene i y_history enn å bruker rad-arrays som skriver over hverandre hvert tidssteg, men bør være litt snillere med CPU'en.

1.5 Oppgave 4

Se Vedlegg 2 for oppgavekoden.

$$y_i^0 \quad = \sin(7\pi \tfrac{i}{N-1}) \quad , \quad 0 \leq i \leq N-1 \tag{O.5}$$

Når $m_0 = m_{N-1} >> m$, får vi følgende uttrykk for y_0 og y_{N-1} fra (1.1) og (1.2):

$$\begin{aligned} y_0^+ \quad &=\sim 0 + 2y_i^0 - y_i^- \quad = 2y_i^0 - y_i^- \\ y_{N-1}^+ \quad &=\sim 0 + 2y_i^0 - y_i^- \quad = 2y_i^0 - y_i^- \end{aligned} \tag{8}$$

Gitt $y^0 = y^-$ vil altså endepunktene få uendret y-posisjon (tilnærmet, ettersom m leddet blir tilnærmet 0 når $m >> m_i$) i neste tidssteg y^+.

$$\begin{aligned} y_0^+ \quad &= y_0^0 \\ y_{N-1}^+ \quad &= y_{N-1}^0 \end{aligned} \tag{9}$$

Initialbetingelsen for y_i^0 kan være angitt som utslag i alle posisjoner ved t = 0. Vil vi da finne det forrige steget, trekker vi fra den tidsderiverte av utslaget i alle posisjoner ved samme tidspunkt. Altså $y_i^- = y_i^0 - \Delta t \frac{dy_i^0}{dt}$. (O.5) gir oss da

$$y_i^- \quad = \sin(7\pi \tfrac{i}{N-1}) - \Delta t \cdot 0 \tag{10}$$

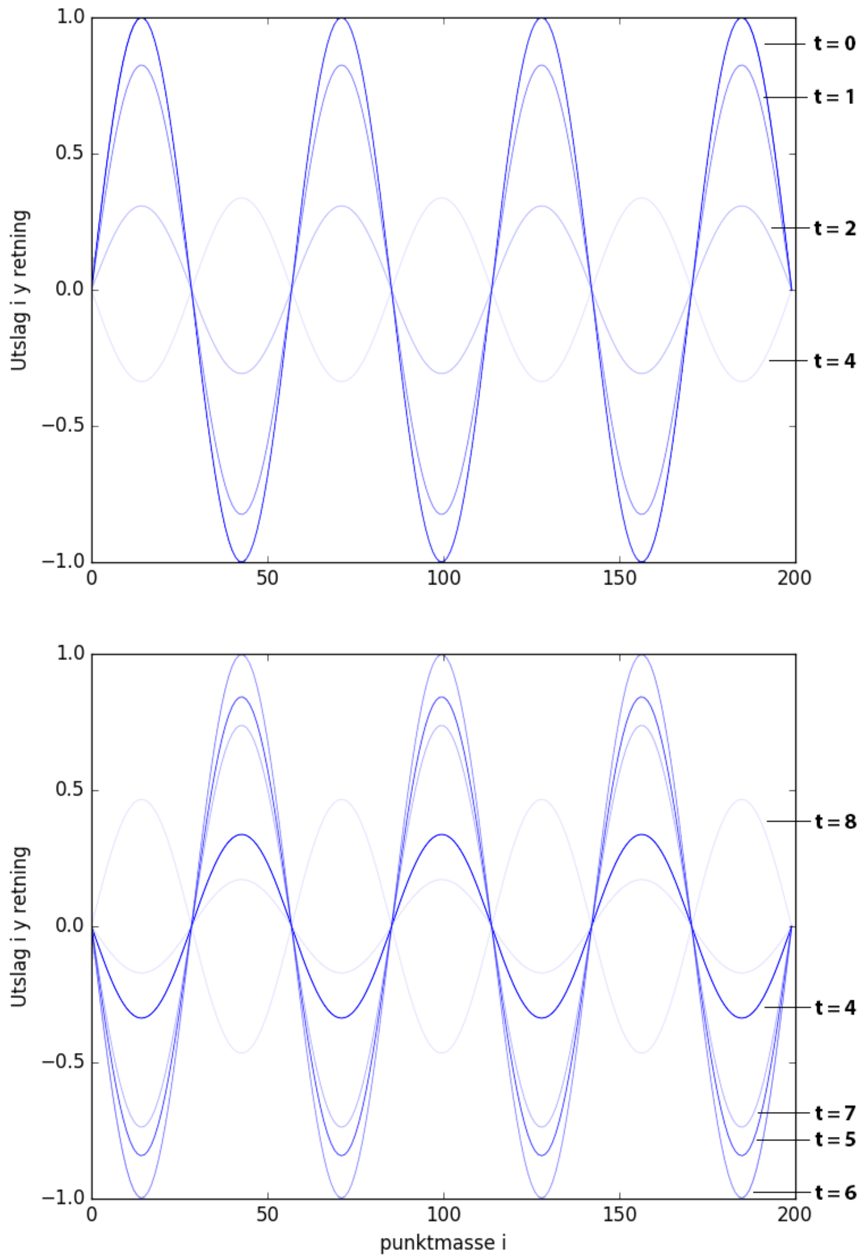
Fordi funksjonen vår ikke er tidsavhengig. Det vil si $\frac{dy_i^0}{dt} = 0$. Vi kan derfor sette $y_i^- = y_i^0$.

Strengen vår har lengden L og initialbetingelsene våre gir oss en streng i form av en sinusfunksjon $\sin(7\pi \frac{i}{L})$. Vi finner bølgelengden til denne funksjonen $\lambda = \frac{7\pi L}{2\pi} \rightarrow L = \frac{7\pi}{2\pi}\lambda = \frac{7}{2}\lambda \rightarrow$

$$\begin{aligned} \sin(kx - \omega t) \quad &= \sin(ki - 0) \\ k \quad &= \tfrac{7\pi}{L} = \tfrac{2\pi}{\lambda} \\ \lambda \quad &= \tfrac{2}{7}L \end{aligned} \tag{11}$$

Som betyr at bølgebevegelsen vår her tilsvarer en stående bølge ($\lambda_n = \frac{2}{n}L$). Kjører vi programmet, får vi denne stående bølgen med forventede noder.[4]

[4]For animasjon og gif, se: http://folk.uio.no/andhel/FYS2130/ eller http://imgur.com/qMSNTrt



Figur 4: I det øverste bilde ser man strengens tidsutvikling i y-retning etter de fire første $(6\Delta t) \cdot t$ intervallene og de neste fire $(6\Delta t) \cdot t$ i det nederste hvor opasiteten reduseres over tid.

1.6 Oppgave 5

Se Vedlegg 3 for oppgavekoden. Fra (11) vet vi at $L = N\Delta x = \frac{7}{2}\lambda$. Bruker vi (5) får vi da ut ifra dette følgende uttrykk

$$\begin{aligned} v_B &= \sqrt{\frac{k}{m}}\Delta x &= \sqrt{\frac{k}{m}}\frac{7\lambda}{2N} \\ f &= \sqrt{\frac{k}{m}}\frac{7}{2N} \end{aligned} \tag{12}$$

Vi vil finne antall tidssteg $n\Delta t$ vi må ta for at programmet har kjørt gjennom 10 perioder $10T = 10\frac{1}{f}$

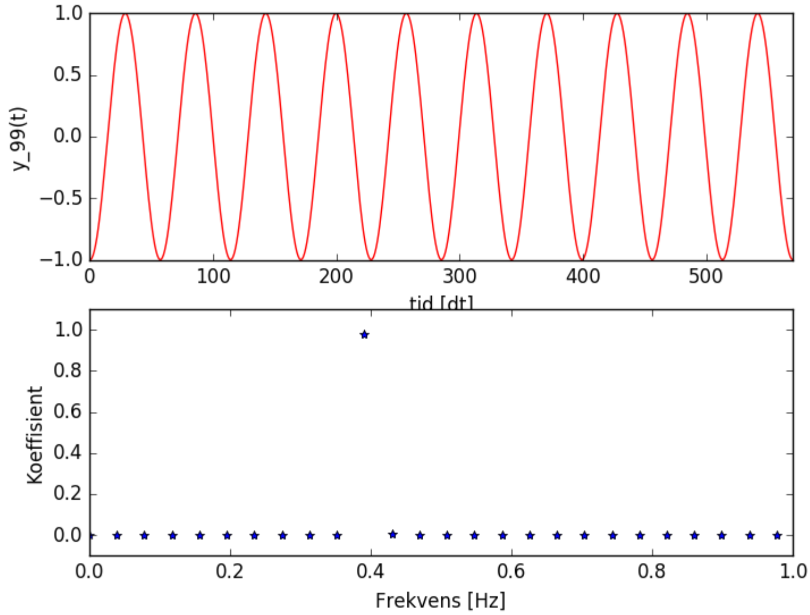
$$\begin{aligned} n\Delta t &= 10(\sqrt{\frac{k}{m}}\frac{7}{2N})^{-1} \\ n &= \sqrt{\frac{m}{k}}\frac{20N}{7\Delta t} \end{aligned} \tag{13}$$

med $\Delta t = \sqrt{\frac{m}{k}} = \sqrt{\frac{0.02}{10}} = 0.0447s$ vil det forventede antall tidssteg bli $n = \frac{20N}{7} = 571.43 \simeq 572$ som gir oss forventet periode $T = \frac{n\Delta x}{10} = 2.55s$ og svingefrekvens $f = \frac{1}{T} = 0.3911s^{-1}$

Lar vi programmet kjøre 10 perioder istedenfor et visst antall tidssteg (ved å telle hvor mange ganger verdi $y_9 9$ går forbi initialverdien sin[5]) får vi $T = 57.2\Delta t = 25.58s$ og $f = 0.03909s^{-1}$.

Dette stemmer nokså bra med både teoretiske verdier og Fast Fourier (se figur 5). Med økt nøyaktighet (flere tidssteg og lavere Δt) ville vi nok kunne fått et resultat som var enda nærmere de forventede verdiene.

[5]Vi kunne også brukt den forventede verdien vi har regnet oss frem til som input, men programmerer man det på denne måten vil programmet finne antall tidssteg per periode selv om man ikke har funnet det analytisk. Se Streng objektet



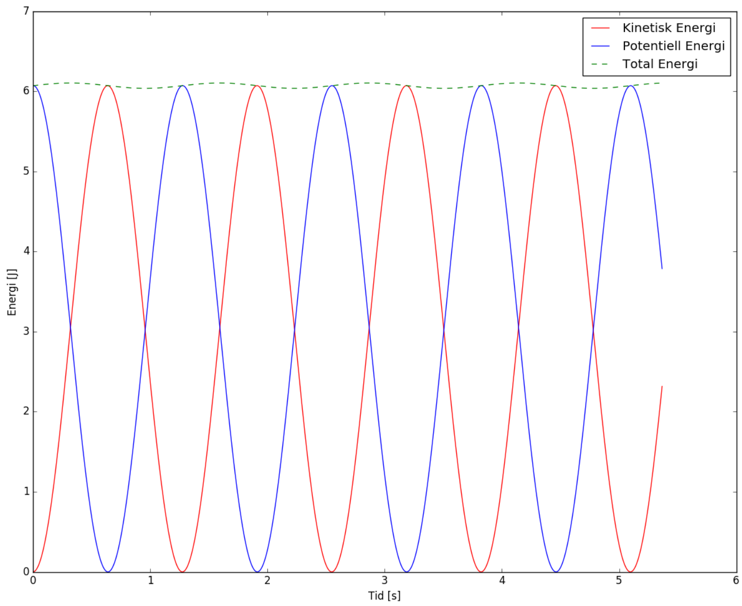
Figur 5: I det øverste bilde ser vi bevegelsen til punktmassen y_{99} over tid. I det nederste bildet ser vi at FFT bekrefter den forventede verdien for svingefrekvensen.

1.7 Oppgave 6

Se Vedlegg 4 for oppgavekoden. For å finne den totale energien i systemet kan vi bruke de kjente formelene (14)

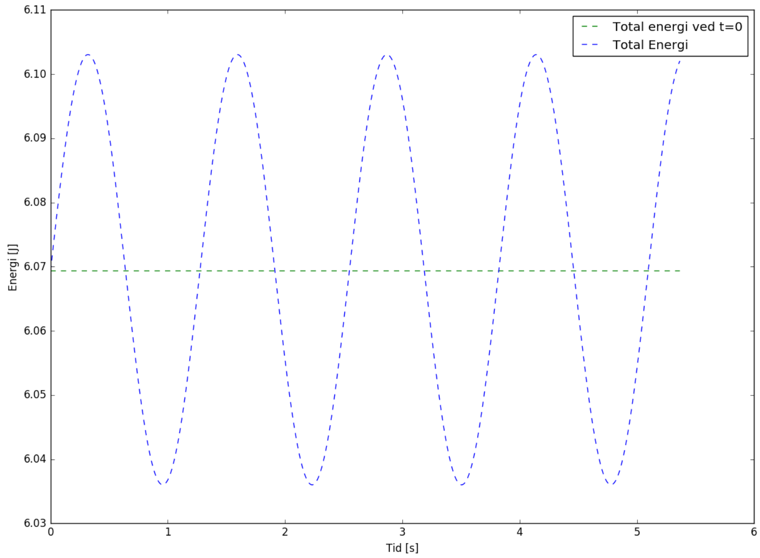
$$\begin{aligned} E_p &= \tfrac{1}{2}k(y_i^0)^2 \\ E_k &= \tfrac{1}{2}m(\tfrac{\Delta y}{\Delta x})^2 \\ E_{tot} &= E_p + E_k \end{aligned} \tag{14}$$

Hvor $\frac{\Delta y}{\Delta x}$ her blir $\frac{y_i^+ - y_i^0}{\Delta x}$. For å bestemme om den totale energien (summen av den kinetiske og potensielle energien langs hele strengen) er bevart i programmet vårt sjekker vi $\sum_0^{N-1} E_{i,tot}$ i det første tidssteget og sammenligner den totale energien i resten av tidsstegene $\sum_0^{N-1} E_{i,tot}^+$ med denne verdien. Energien er bevart dersom verdien ikke endrer seg over tid.



Figur 6: Vi ser her at E_{tot} holder seg nokså konstant, men har et lite avvik. Se 7

Øker man nøyaktigheten ved å redusere $\Delta t \to 0$ vil avviket fra den orginale totalenergien $\Delta E_{tot} \to 0$. Fra dette kan vi konkludere med at totalenergien

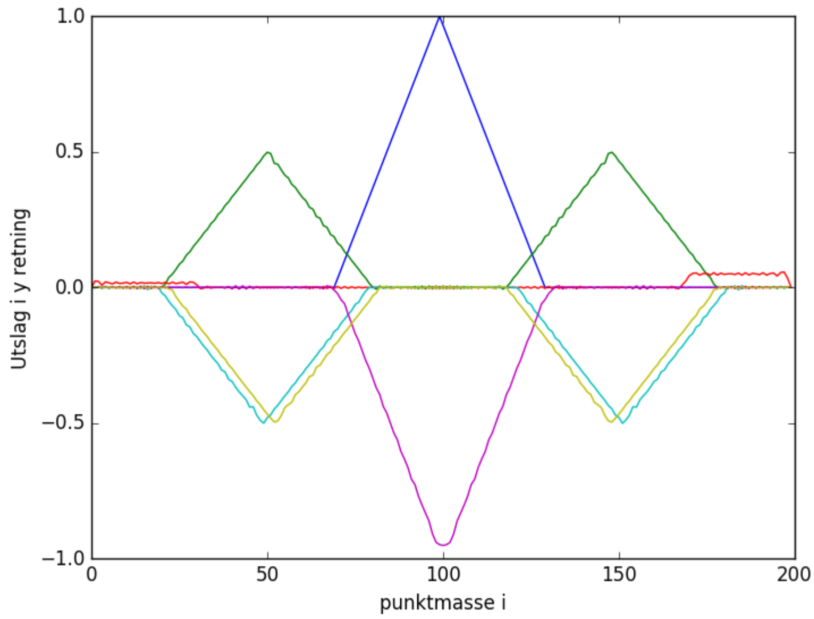


Figur 7: Vi ser her at den totale energien over tid avviker fra den opprinnelige E_{tot}, men likevektspunktet ligger rundt den orginale verdien (reduserer man Δt vil $< E_{tot} > \to E_{tot}$ ved t=0).

er bevart i denne numeriske løsningen.

1.8 Oppgave 7

Se Vedlegg 5 for oppgavekoden. Initialbetingelsen for y_i^0 er angitt som utslag i alle posisjoner ved t = 0. Vi 'slipper' strengen i y^0 med det oppgitte, trekantede initial-utslage uten tidligere bevegelse. Vi setter derfor $y^- = y^0$.



Figur 8: Som man vil se i animasjonen, går strengen fra initialutslaget til de 2 mindre trekantene i retning kantene for så å bli reflektert og sendt i motsatt retning til de møtes i midten, former den reflekterte trekanten og (på grunn av symmetri) fortsette med utslag til høyre lik -(utslag til venstre)

Ettersom initialutslaget er sentrert og symmetrisk får vi (som forventet) to like og motsatte rette bølger som strekker seg ut fra midten. Når bølgen møter endepunktene, gir impedansen fullstendig reflekterte bølger og utslaget får motsatt fortegn av de inkommende bølgene på grunn av de reflekterende randbetingelsene $m_0 = m_{N-1} >> m$.[6]

[6]Jeg beklager for de undervelmende figurene i denne og de neste animasjonsoppgavene. Den orginale planen ble for ambisiøs, så de ble en raskt utskrift i siste liten med grunnleggende plot-kode

1.9 Oppgave 8

Se Vedlegg 6 for oppgavekoden.

Hvis bølgen beveger seg til høyre, vil det si at i det forrige tidssteg vil den befinne seg til venstre for det nåværende tidssteget. For hvert steg Δt vil trekanten befinne seg i et punktmasse-intervall $\Delta trekant^{-} < \Delta trekant^{0}$.

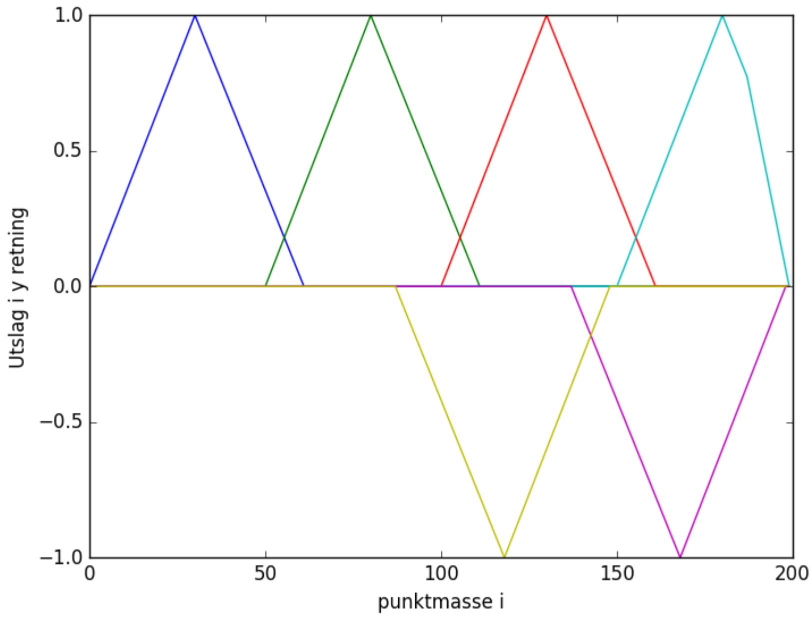
$$\begin{aligned} y_i^- &= y_i^0 - \Delta x \tfrac{dy_i^0}{dx} = y_i^0 - \Delta x \tfrac{y_i^0 - y_{i-1}^0}{\Delta x} \\ y_i^- &= y_i^0 - (y_i^0 - y_{i-1}^0) \\ y_i^- &= y_{i-1}^0 \end{aligned} \tag{15}$$

For å finne hvor lange tidsstegene skal være for at (15) er tilfelle (altså, $0 \leq c \leq 1$ fra (6)). bruker vi (5) for å se på

$$\begin{aligned} y(x_1, t) &= y(x_0 + vt) \\ vt = v_B \Delta t &= \sqrt{\tfrac{k}{m}} \Delta x \sqrt{\tfrac{m}{k}} c \\ v_B &= \tfrac{\Delta x}{\Delta t} c \end{aligned} \tag{16}$$

Fordi $v_B \leq \frac{\Delta x}{\Delta t}$ (6), må dermed $c = 1$ og vi får at $\Delta t = \sqrt{\frac{m}{k}}$ må være tilfelle hvis vi vil ha oppgitt bølgebevegelse [7]. Se Figur 9 for en demonstrasjon av programmet.

[7] En anologi man kan bruke vil være at punktmasse-strengen vår er for pixelert for at en lavere Δt vil fungere



Figur 9: Det trekantede utslaget går fra blå → grønn → rød osv. Altså fikk vi en bevart bevegelse mot høyre. Med gitte randbetingelser blir den så refletert når den treffer kanten.

1.10 Oppgave 9

Se Vedlegg 7 for oppgavekoden.

Med det oppgitte masseforholdet vil det forventede impedanseforholdet bli

$$\frac{Z_i}{Z_j} = \frac{\sqrt{mk}}{\sqrt{3mk}} = \frac{1}{\sqrt{3}} = 0.577 \tag{17}$$

Fra dette og en inkommende bølgeamplitude $A_i = 1$ vil forventede transmitert og reflektert utslag bli følgende (vi er ute etter forholdet, så m og k kan her setter lik 1 for å gjøre det enklere og penere)

$$\begin{aligned} A_r &= \frac{Z_i - Z_i}{Z_i + Z_j} &= \frac{1-\sqrt{3}}{1+\sqrt{3}} &= \sqrt{3} - 2 &= -0.268 \\ A_t &= \frac{2Z_i}{Z_i + Z_j} &= \frac{2}{1+\sqrt{3}} &= \sqrt{3} - 1 &= 0.732 \end{aligned} \tag{18}$$

Vi finner dermed den reflekterte og transmiterte amplituden numerisk. Ved å finne toppunkt og bunnpunkt i et tidssteg etter bølgen har kollidert med impedansforskjellspunktet, gav programmet vårt følgende Amplituder [8]

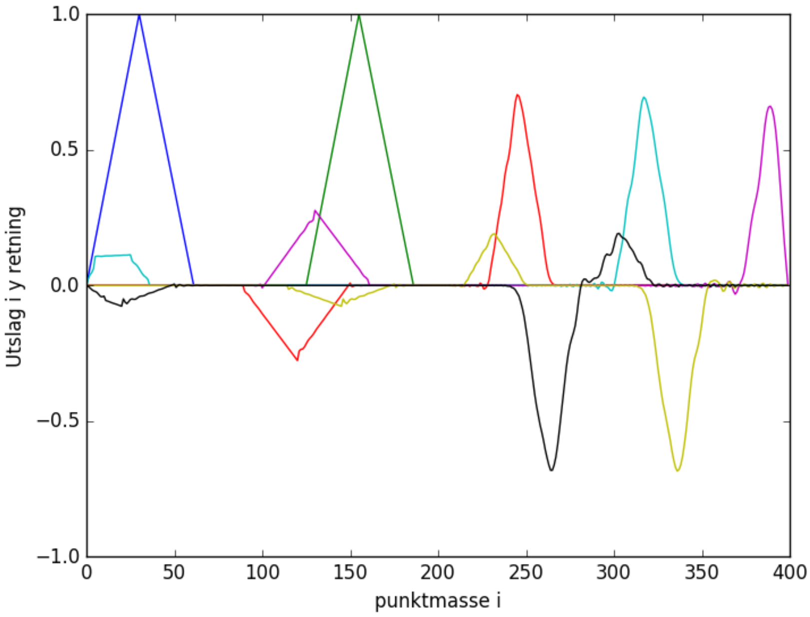
$$\begin{aligned} A_r &= -0.27668576508 \\ A_t &= 0.697144116036 \\ \\ |A_r| + |A_t| &= 0.973829881116 \\ \frac{A_r}{A_t} &= 0.396884602072 \end{aligned} \tag{19}$$

Fra dette kan finne vi impedanseforholdet mellom de to delene av strengen numerisk.

[8] Det totale utslaget $|A_r| + |A_t|$ ville vi forventet at skulle være lik den inkommende amplituden (1.0), men på grunn av numeriske feil ble den ikke bevart her.

$$
\begin{aligned}
A_t &= \tfrac{2Z_i}{Z_i+Z_j} \\
(Z_i + Z_j)A_t &= 2Z_i \\
Z_iA_t - 2Z_i &= -Z_jA_t \\
Z_i &= \tfrac{-Z_jA_t}{A_t-2} \\
\tfrac{Z_i}{Z_j} &= \tfrac{A_t}{2-A_t} \\
\tfrac{Z_i}{Z_j} &= \tfrac{0.6971}{2-0.6971} = 0.535
\end{aligned}
\tag{18}
$$

Vi får en verdi som er nokså lik den forventede verdien, men med et lite avvik. Kjører vi programmet og ser på animasjonen av strengen over tid får vi et lite hakk i det initielle refleksjonsutslaget, som sannsynligvis kommer fra numeriske feil. Dette kan være årsaken til forskjellen fra det forventede impedanseforholdet.



Figur 10: utslaget går fra blå → grønn → rød osv. Ettersom tiden går og mange utslag treffer masseforskjellspunktet blir nye refleksjoner skapt og bølgebevegelsen blir seende nokså kaotisk ut.

Appendices

.1 Lenker

- Alle relevante filer til prosjektoppgaven
- Oppgave 4 python kode
- Oppgave 4 animasjon mp4
- Oppgave 4 animasjon GIF
- Oppgave 5 python kode
- Oppgave 6 python kode
- Oppgave 7 python kode
- Oppgave 7 animasjon mp4
- Oppgave 7 animasjon GIF
- Oppgave 8 python kode
- Oppgave 8 animasjon mp4
- Oppgave 8 animasjon GIF
- Oppgave 9 python kode
- Oppgave 9 animasjon mp4
- Oppgave 9 animasjon GIF

.2 Python kode

breakatwhitespace

```
import numpy as np

class streng():

    def __init__(self, m, k, N, I, dt, y_minus, y_0):
        self.y = np.zeros(N)      # current y-positions
        self.m = np.zeros(N)      # point masses
        self.k = np.zeros((N)-1) # springs between points of mass
        self.dt = dt              # Time interval
        self.t = I*self.dt        # Total time of all iterations

        #Convert simple initial conditions to specified conditions if proper values have been assigned
        if (not isinstance(m, float)) and len(m) == len(self.m):            # if m is an array of numbers
            for i in range(0, len(self.m)):
                self.m[i] = m[i]
        elif isinstance(m, float):           # if m is a single value, assign it to all point masses
            for i in range(0, len(self.m)):
                self.m[i] = m
        if (not isinstance(k, float)) and len(k) == range(len(self.k)):      # if k is an array of numbers
            for i in range(0, len(self.k)):
                self.k[i] = k[i]
        elif isinstance(k, float):           # if k is a single value, assign it to all spring constants
```

```
            for i in range(0, len(self.k)):
                self.k[i] = k

        # initialize arrays for saving data for each point in time
        if I > 1:
            self.y_history = np.zeros((I, N))  # Store all I iterations  for each point mass N
            self.y_history[0] = y_minus         # initial condition:  first y-positions (y_minus)
            self.y_history[1] = y_0             # initial condition: second set of y-positions (y_zero)
        else:                                   # If we don't know how many iterations we are going to run through
            self.y_history = np.array(y_minus)  # make room for the two initial conditions
            self.y_history = np.vstack((self.y_history, np.array(y_0)))


    # execute movement of point masses over time
    def __call__(self, boundary, P, x):
        i_max = len(self.y_history[0])-1

        if boundary == True:                    # add weights M >> m at the ends if we want a bound string
            self.m[0] = 1000000000.0*self.m[0]
            self.m[len(self.m)-1] = 1000000000.0*self.m[len(self.m)-1]

        if P == 0:                              # if you have a predetermined number of iterations
            # Find y values over time for all point masses
            for t in range(2,len(self.y_history)):      # N times, every dt time-interval (except the two initial conditions)
                #handle boundaries - at 0, force from the left = 0, at N-1, force from the right = 0
                self.y_history[t][0] = (self.ForceRight(0,t-1)/self.m[0])*self.dt**2 + 2*self.y_history[t-1][i_max] -
        self.y_history[t-2][i_max]
                self.y_history[t][i_max] = (self.ForceLeft(i_max,t-1)/self.m[len(self.m)-1])*self.dt**2 +
        2*self.y_history[t-1][i_max] - self.y_history[t-2][i_max]
                # non-boundary values
                for i in range(1, i_max):                       # For every point mass i find its y position
                    self.y_history[t][i] = self.y_next(i, t-1)  # t-1 = y0 time interval, t = y+

        elif P != 0:        # if you want the string to move a certain number of periods P  [oppgave 5, P=10, x=99]
            t, periods = 2,0
            # calculate new y-values until position x has passed its initial position P times
            while periods < P:
                # non-boundary values
                for i in range(1, i_max):               # For every point mass i find its y position
                    self.y[i] = self.y_next(i, t-1)     # t-1 = y0 time interval, t = y+

                #handle boundaries - at 0, force from the left = 0, at N-1, force from the right = 0
                self.y[0] = (self.ForceRight(0,t-1)/self.m[0])*self.dt**2 + 2*self.y_history[t-1][i_max] -
        self.y_history[t-2][i_max]
                self.y[i_max] = (self.ForceLeft(i_max,t-1)/self.m[len(self.m)-1])*self.dt**2 +
        2*self.y_history[t-1][i_max] - self.y_history[t-2][i_max]
                # check if position x of the string has traveled a period (period++ if so)
                if (self.y[x] <= self.y_history[0][x] and self.y[x-1] >= self.y_history[0][x]) or (self.y[x] >=
        self.y_history[0][x] and self.y[x-1] <= self.y_history[0][x]):
                    periods = periods +1
                self.y_history = np.vstack((self.y_history, np.array(self.y)))
                t = t+1
            self.t = (t+2)*self.dt                  # Store the total time the string has been in motion


    # returns y-position in the next time interval y+_i
    def y_next(self, i,t):
        return (self.Force(i,t)/self.m[i])*self.dt**2 + 2*self.y_history[t][i] - self.y_history[t-1][i]

    #returns Force from the left spring
    def ForceLeft(self, i, t):  #where t is the current time interval we're dealing with and i is position along the string
        return -self.k[i-1]*(self.y_history[t][i]-self.y_history[t][i-1])

    #returns force from the right spring
    def ForceRight(self, i, t):     #where t is the current time interval we're dealing with and i is position along the
        string
        return -self.k[i]*(self.y_history[t][i]-self.y_history[t][i+1])

    #returns total force from springs
    def Force(self,i,t):
        return (self.ForceRight(i, t) + self.ForceLeft(i, t))
```

Listing 1: Streng Objekt kode

breakatwhitespace

```
import numpy as np
import matplotlib.pyplot as plt
import sys
import matplotlib.animation as animation
from Streng import *
```

```
# Initial conditions
m           = 0.02                  # point masses
Iterations  = 1200                  # number of iterations
N           = 200                   # number of x value (how far the string stretches)
k           = 10.0                  # spring constant
dt          = 1*np.sqrt(m/k)        # time interval

y_minus = np.zeros(N)        #y-positions in first iteration (initial condition)
y_0 = np.zeros(N)            #y-positions in second iteration (initial condition)
for i in range(1, N-1):
    y_0[i] =  np.sin((7.0*np.pi*i)/float(N-1.0))
    y_minus[i] = y_0[i]

# Create string by sending initial conditions to Streng object
S = streng(m, k, N, Iterations, dt, y_minus, y_0)
S(True,0,0)                                  # Calculate movement of bound=true string



#simple animation code (Functions based on matplotlib.org example)
fig, ax = plt.subplots()
x = np.linspace(0,N-1,N)
line, = ax.plot(x, S.y_history[0])
ax.set_xlabel("punktmasse i")
ax.set_ylabel("Utslag i y retning")
ax.set_ylim([-1,1])
ax.set_xlim([0,N])
ax.set_title('Streng med sinusfunksjon som initialutslag')

def animate(l):
    line.set_ydata(S.y_history[l])
    return line,

def init():
    line.set_ydata(np.ma.array(x, mask=True))
    return line,

#print and save plot
def savePlot(filename, frame):
    fig2, ax2 = plt.subplots()
    ax2.plot(x, S.y_history[frame])
    ax2.set_xlabel("punktmasse i")
    ax2.set_ylabel("Utslag i y retning")
    ax2.set_ylim([-1,1])
    ax2.set_xlim([0,N])
    fig2.savefig(str(filename)+'.png')

plt.rcParams['animation.ffmpeg_path'] = 'C:\\FFmpeg\\bin\\ffmpeg.exe' #path for windows where you've placed
       FFmpeg
FFwriter = animation.FFMpegWriter(fps=30, extra_args=['-vcodec', 'libx264'])
ani = animation.FuncAnimation(fig, animate, np.arange(0, Iterations), init_func=init, interval=25, blit=True)
plt.show()                                   #show string movement
for i in range(0,61,6):
    savePlot("Oppgave4_plot_frame"+str(i),i)           #store 11 images of one period
ani.save('Oppgave4_animation.mp4', writer = FFwriter)    #save mp4 animation file



#quick note. I had an accident where i tried to rename some variables in one file, and due
#to how eclipse works, the variables got renamed in all files. I think the problem has been
#fixed, but there could be stray variables with the wrong name (l and k)
```

Listing 2: Oppgave 4 kode

breakatwhitespace

```
import numpy as np
import matplotlib.pyplot as plt
from Streng import *

# Initial conditions
m           = 0.02                  #masses
Iterations  = 0                     #There is no predetermined number of iterations
N           = 200                   #number of x value (how far the string stretches)
k           = 10.0                  #spring constant
dt          = 1*np.sqrt(m/k)        #time interval

y_minus = np.zeros(N)        #y-positions in first iteration (initial condition)
y_0 = np.zeros(N)            #y-positions in second iteration (initial condition)
for i in range(0, N-1):
    y_0[i] = y_minus[i] = np.sin(7.0*np.pi*(i/(N-1.0)))     #np.sin(7.0*np.pi*(i/(N-1.0)))     # y- = y0
```

```
# Create string by sending initial conditions to Streng object
S = streng(m, k, N, Iterations, dt, y_minus, y_0)
S(True,10,99)                                   # Calculate movement until point 99 has traveled 10 periods



# initialize Fourier & time plot values
t = np.linspace(0,len(S.y_history),len(S.y_history))
y_99 = np.zeros(len(S.y_history))
T = int(S.t/S.dt) #10 x period
print "10*T =",T,"dt =",S.t,"sekund"
print "f =",1/S.t
for i in range(0,len(S.y_history)):
    y_99[i] = S.y_history[i][99]        #extract y_99 values

#Fourier
F = np.fft.fft(y_99,T)/T
freq = np.fft.fftfreq(T, d=S.dt)
freq = np.abs(freq[:T])
F = (2*np.abs(F[:T]))**2

# Function to create multiple plots in one
def multiPlot(t, y, ylabel, xlabel,y_axes,x_axes, P, LineAppearance):
    ax2 = fig.add_subplot(P[0],P[1],P[2])
    ax2.set_ylabel(ylabel)
    ax2.set_xlabel(xlabel)
    ax2.set_ylim(y_axes)
    ax2.set_xlim(x_axes)
    ax2.plot(t, y, LineAppearance)

#print and save plot
def savePlot(filename):
    plt.show()
    fig.savefig(str(filename)+'.png')

fig = plt.figure()      #Shared figure for plots
multiPlot(t, y_99, "y_99(t)", "tid [dt]",[-1,1],[0,len(S.y_history)],[2,1,1],'r') #y(t)
multiPlot(freq, F, "Koeffisient", "Frekvens [Hz]",[-0.1,1.1],[0,1],[2,1,2],'b*')  #fft
savePlot("Oppgave5_plot")
```

Listing 3: Oppgave 5 kode

breakatwhitespace

```
import numpy as np
from math import pi,sin,sqrt
import matplotlib.pyplot as plt
from Streng import *

# Initial conditions
m          = 0.02                 #masses
Iterations  = 4200                 #number of iterations
N          = 200                  #number of x value (how far the string stretches)
k          = 10.0                 #spring constant
dt         = 0.01*sqrt(m/k)          #time interval

y_minus = np.zeros(N)        #y-positions in first iteration (initial condition)
y_0 = np.zeros(N)            #y-positions in second iteration (initial condition)
for i in range(0, N-1):
    y_0[i] = y_minus[i] = sin(7.0*pi*(i/(N-1.0)))  # y- = y0



# Create string by sending initial conditions to Streng object
S = streng(m, k, N, Iterations, dt, y_minus, y_0)
S(True,0,0)                                    # Calculate movement of bound=true string

#Find energies
E, E2 = np.zeros((Iterations-1, N)),np.zeros(Iterations-1)
E_k, E_k2 = np.zeros((Iterations-1, N)),np.zeros(Iterations-1)
E_p, E_p2 = np.zeros((Iterations-1, N)),np.zeros(Iterations-1)
E_average = 0

for l in range (0,Iterations-1):    # for every iteration
    for i in range(0, N-1):         # in every point along the string
        E_k[l][i]  = 0.5*m*(float(S.y_history[l+1][i]-S.y_history[l][i])/dt)**2 #store kinetic energy in point i at time l
        E_p[l][i]  = 0.5*k*(S.y_history[l][i+1]-S.y_history[l][i])**2          #store potential energy in point i at time l
        E[l][i]   = E_k[l][i] + E_p[l][i]                                      #store total energy in point i at time l

    E_k2[l] = sum(E_k[l])           # Total kinetic energy in all point-masses in this iteration
```

```
    E_p2[l] = sum(E_p[l])           # Total potential energy in all springs in this iteration
    E2[l] = E_k2[l] + E_p2[l]       # stotal energy of the system this iteration
    E_average = E_average + E2[l]   # add up all total energies
E_average = E_average/Iterations    # find average energy of every point in time


# plot kinetic, potential and total energy over time
t = np.linspace(0,(Iterations)*dt,Iterations-1)   #time intervals for plot
plt.xlabel("Tid [s]")
plt.ylabel("Energi [J]")
plt.plot(t,E_k2,'r-',t,E_p2,'b',t,E2,'g--')
plt.legend(["Kinetisk Energi", "Potentiell Energi", "Total Energi"])
plt.show()

# plot deviation from the average total energy over time
average = np.linspace(E_average,E_average,Iterations-1)
Initial_Energy = np.linspace(E2[0],E2[0],Iterations-1)
plt.xlabel("Tid [s]")
plt.ylabel("Energi [J]")
plt.plot(t,Initial_Energy,'g--',t,E2,'b--') #t,average,'r-',
plt.legend(["Total energi ved t=0", "Total Energi"]) #"gjennomsnittlig Energi",
plt.show()
```

Listing 4: Oppgave 6 kode

breakatwhitespace

```
import numpy as np
import matplotlib.pyplot as plt
import sys
import matplotlib.animation as animation
from Streng import *

# Initial conditions
m          = 0.02               #masses
Iterations  = 2500              #number of iterations
N          = 200                #number of x value (how far the string stretches)
k          = 10.0               #spring constant
dt         = 0.99*np.sqrt(m/k)  #time interval

y_minus = np.zeros(N)       #y-positions in first iteration (initial condition)
y_0 = np.zeros(N)           #y-positions in second iteration (initial condition)
for i in range(0, N-1):     #fill initial iterations for desired wave function
    if (70 <= i and i <= 99):
        y_minus[i]  = (i-69.0)/30.0
        y_0[i]      = (i-69.0)/30.0
    elif (100 <= i and i <= 128):
        y_minus[i]  = (129.0-i)/30.0
        y_0[i]      = (129.0-i)/30.0
    else:
        y_minus[i]  = 0
        y_0[i]      = 0



# Create string by sending initial conditions to Streng object
S = streng(m, k, N, Iterations, dt, y_minus, y_0)
S(True,0,0)                                 # Calculate movement of bound=true string



#simple animation code (Functions based on matplotlib.org example)
fig, ax = plt.subplots()
x = np.linspace(0,N-1,N)
line, = ax.plot(x, S.y_history[0])
ax.set_xlabel("punktmasse i")
ax.set_ylabel("y posisjon")
ax.set_ylim([-1,1])
ax.set_xlim([0,N])
ax.set_title('Sentrert trekantet initialutslag ')

def animate(l):
    line.set_ydata(S.y_history[l])
    return line,

def init():
    line.set_ydata(np.ma.array(x, mask=True))
    return line,
```

```
plt.rcParams['animation.ffmpeg_path'] = 'C:\\FFmpeg\\bin\\ffmpeg.exe' #path for windows where you've placed
        FFmpeg
FFwriter = animation.FFMpegWriter(fps=30, extra_args=['-vcodec', 'libx264'])
ani = animation.FuncAnimation(fig, animate, np.arange(0, Iterations), init_func=init, interval=10, blit=True)
        #increase interval to slow down animation
fig2, ax2 = plt.subplots()
for i in range(0,300,50):
    ax2.plot(x, S.y_history[i])
    ax2.set_xlabel("punktmasse i")
    ax2.set_ylabel("Utslag i y retning")
    ax2.set_ylim([-1,1])
    ax2.set_xlim([0,N])
fig2.savefig("Oppgave7_plot2_frame"+'.png')

plt.show()
ani.save('Oppgave7_animation.mp4', writer = FFwriter)
```

Listing 5: Oppgave 7 kode

breakatwhitespace

```
import numpy as np
import matplotlib.pyplot as plt
import sys
import matplotlib.animation as animation
from Streng import *

# Initial conditions
m           = 0.02                  #masses
Iterations  = 1200                  #number of iterations
N           = 200                   #number of x value (how far the string stretches)
k           = 10.0                  #spring constant
dt          = 1*np.sqrt(m/k)        #time interval

y_minus = np.zeros(N)        #y-positions in first iteration (initial condition)
y_0 = np.zeros(N)            #y-positions in second iteration (initial condition)
for i in range(0, N-1):      #fill initial iterations for desired wave function
    if (1 <= i and i <= 30):
        y_0[i]       = i/30.0
        y_minus[i]   = y_0[i-1]  #place y-[i] in the previous position of y0[i]
    elif (31 <= i and i <= 61):
        y_0[i]       = (61-i)/31.0
        y_minus[i]   = y_0[i-1]
    else:
        y_0[i]       = 0
        y_0[i-1]     = y_0[i]



# Create string by sending initial conditions to Streng object
S = streng(m, k, N, Iterations, dt, y_0, y_minus)
S(True,0,0)                                        # Calculate movement of bound=true string



#simple animation code (Functions based on matplotlib.org example)
fig, ax = plt.subplots()
x = np.linspace(0,N-1,N)
line, = ax.plot(x, S.y_history[0])
ax.set_xlabel("Punktmasse i")
ax.set_ylabel("Y Posisjon")
ax.set_ylim([-1,1])
ax.set_xlim([0,N])
ax.set_title('Trekantet initialutslag med bevegelse mot hoyre')

def animate(l):
    line.set_ydata(S.y_history[l])
    return line,

def init():
    line.set_ydata(np.ma.array(x, mask=True))
    return line,

plt.rcParams['animation.ffmpeg_path'] = 'C:\\FFmpeg\\bin\\ffmpeg.exe' #path for windows where you've placed
        FFmpeg
FFwriter = animation.FFMpegWriter(fps=30, extra_args=['-vcodec', 'libx264'])
ani = animation.FuncAnimation(fig, animate, np.arange(0, Iterations), init_func=init, interval=10, blit=True)

fig2, ax2 = plt.subplots()
for i in range(0,300,50):
    ax2.plot(x, S.y_history[i])
    ax2.set_xlabel("punktmasse i")
```

```
    ax2.set_ylabel("Utslag i y retning")
    ax2.set_ylim([-1,1])
    ax2.set_xlim([0,N])
fig2.savefig("Oppgave7_plot2_frame"+'.png')


plt.show()
ani.save('Oppgave8_animation.mp4', writer = FFwriter)
```

Listing 6: Oppgave 8 kode

breakatwhitespace

```
import numpy as np
import matplotlib.pyplot as plt
import sys
import matplotlib.animation as animation
from Streng import *
from sympy.physics.quantum.tests.test_sho1d import a_rep

# Initial conditions
N           = 400                  #number of x value (how far the string stretches)
m           = np.zeros(N)          #masses
Iterations  = 2400                 #number of iterations
k           = 10.0                 #spring constant
dt          = 1*np.sqrt(0.02/k)    #time interval with lowest mass value

y_minus = np.zeros(N)        #y-positions in first iteration (initial condition)
y_0 = np.zeros(N)            #y-positions in second iteration (initial condition)

for i in range(0, N-1):      #fill initial iterations for desired wave function
    if (1 <= i and i <= 30):
        y_0[i]      = i/30.0
        y_minus[i]  = y_0[i-1]
    elif (31 <= i and i <= 61):
        y_0[i]      = (61-i)/31.0
        y_minus[i]  = y_0[i-1]
    else:
        y_0[i]      = 0
        y_0[i-1]    = y_0[i]

for i in range(0, N):        #add two different masses to the first and last N/2 elements
    if i < N/2:
        m[i] = 0.02          #low mass m for the first 200
    else:
        m[i] = 0.06          #high mass 3*m for the next 200



# Create string by sending initial conditions to Streng object
S = streng(m, k, N, Iterations, dt, y_0, y_minus)
S(True,0,0)                                  # Calculate movement of bound=true string


A_r = 0 #reflected amplitude
A_t = 0 #transmitted amplitude
#finn A_r og A_t
for j in range(0, len(S.y_history[350])-1):
    if S.y_history[350][j] <= A_r:
        A_r = S.y_history[350][j]
    if S.y_history[350][j] >= A_t:
        A_t = S.y_history[350][j]


print "Reflektert A_r:",A_r
print "transmitert A_t:",A_t                 #
print "A_r + A_t = ",np.abs(A_r) +np.abs(A_t)    #total after
print "A_r / A_t = ",np.abs(A_r)/np.abs(A_t)     #ratio

#simple animation code (Functions based on matplotlib.org example)
fig, ax = plt.subplots()
x = np.linspace(0,N-1,N)
line, = ax.plot(x, S.y_history[0])
ax.set_xlabel("punktmasse i")
ax.set_ylabel("y posisjon")
ax.set_ylim([-1,1])
ax.set_xlim([0,N])
ax.set_title('Strengbevegelse hvor punktmassene ikke er like')

def animate(l):
    line.set_ydata(S.y_history[l])
    return line,
```

```
def init():
    line.set_ydata(np.ma.array(x, mask=True))
    return line,

plt.rcParams['animation.ffmpeg_path'] = 'C:\\FFmpeg\\bin\\ffmpeg.exe' #path for windows where you've placed
      FFmpeg
FFwriter = animation.FFMpegWriter(fps=30, extra_args=['-vcodec', 'libx264'])
ani = animation.FuncAnimation(fig, animate, np.arange(0, Iterations), init_func=init, interval=10, blit=True)

fig2, ax2 = plt.subplots()
for i in range(0,800,125):
    ax2.plot(x, S.y_history[i])
    ax2.set_xlabel("punktmasse i")
    ax2.set_ylabel("Utslag i y retning")
    ax2.set_ylim([-1,1])
    ax2.set_xlim([0,N])
fig2.savefig("Oppgave9_plot"+'.png')



plt.show()
ani.save('Oppgave9_animation.mp4', writer = FFwriter)
```

Listing 7: Oppgave 9 kode