Wibrująca struna

Weronika Smagór, Aleksandra Motor, Patryk Polczyk
Fizyka Techniczna III rok
Wydział Inżynierii Materiałowej i Fizyki Politechniki Krakowskiej

Luty 2021
Kraków

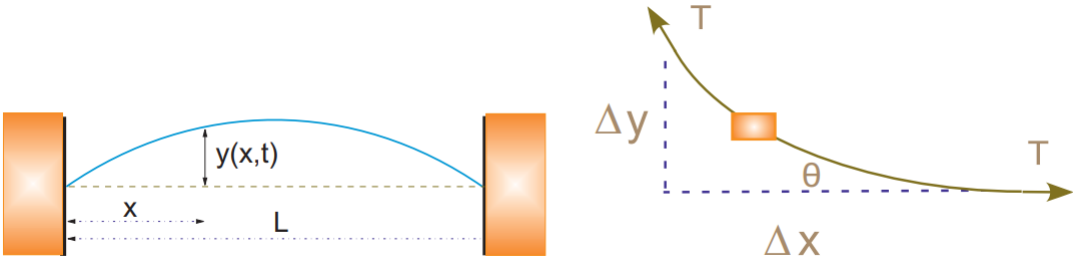
Spis treści

1 Wstęp

Głównym zadaniem naszego projektu jest opracowanie modelu propagacji fal na strunie. Wobec tego rozważamy strunę przywiązaną na obu końcach. Natępnie struna jest szarpana w jednym miejscu, w celu obserwacji pulsu przemieszczającego się wzdłuż struny.

Gdyby tylko jeden koniec struny był uwiązany, podczas gdy będziemy potrząsać wolnym końcem, utworzony zostanie wzór fali stojącej, w którym węzły pozostają na miejscu, a anty-węzły poruszają się tylko w górę i w dół.

2 Równanie fal hiperbolicznych



Do naszych rozważań użyjemy struny o długości L, którą przyczepimy do dwóch stałych punktów. Linka posiada stałą gęstość ρ oraz naprężenie T. W badanym modelu pomijamy siły oporu ruchu. Ruch odbywa się natomiast wzdłuż osi Y, a wartość tego wychylenia położenia zależy od położenia na osi X oraz czasu t – y(x, t). Zakładamy także, że przemieszczenie $\frac{y(x,t)}{L}$ oraz wychylenie $\frac{\partial y}{\partial x}$ są małe. Gdyby wyodrębnić nieskończenie mały odcinek Δx moglibyśmy zauważyć, że różnica składowej pionowego naprężenia tworzy tzw. siłę przywracającą. Siła ta działa na ten odcinek przyspieszająco. Co możemy zapisać za pomocą praw Newtona w następujący sposób:

$$\begin{aligned}\sum F_y &= \rho \Delta x \frac{\partial^2 y}{\partial t^2} \\ &= T sin\Theta(x+\Delta x) - T sin\Theta(x) \\ &= T\frac{\partial y}{\partial x}\bigg|_{x+\Delta x} - T\frac{\partial y}{\partial x}\bigg|_{x} \simeq T\frac{\partial^2 y}{\partial x^2}, \\ \implies \frac{\partial^2 y(x,t)}{2} &= \frac{1}{c^2}\frac{\partial^2 y(x,t)}{2}, \quad c = \sqrt{\frac{T}{\rho}}\end{aligned} \tag{1}$$

Zgodnie z tym co założyliśmy na początku wychylenia są małe, z tego wynika, że $sin = tg = \frac{\partial y}{\partial x}$. Stała c jest prędkością zależną od przyjętych parametrów początkowych, a dokładniej naprężenia T oraz gęstości ρ. Zaznaczyć jednak trzeba, że prędkość c nie jest tożsama z prędkością elementu struny $\frac{\partial y}{\partial x}$.

Do modelowania tego zjawiska zaimplementowaliśmy następujące wartości stałych:

```
#Parametry
rho = 0.01
ten = 40
c = sqrt(ten/rho)
c1 = c
ratio = c*c/(c1*c1)
```

Rysunek 1: Parametry symulacji

Do parametru c1 oraz ratio wrócimy jeszcze w dalszej części omówienia.

$$\frac{y(x,t=0)}{0.005} = \begin{cases} 0, & 0.0 \leq x \leq 0.1, \\ 10x-1, & 0.1 \leq x \leq 0.2, \\ -10x+3, & 0.2 \leq x \leq 0.3, \\ 0, & 0.3 \leq x \leq 0.7, \\ 10x-7, & 0.7 \leq x \leq 0.8, \\ -10x+9, & 0.8 \leq x \leq 0.9, \\ 0, & 0.9 \leq x \leq 1.0 \end{cases}$$

Rysunek 2: Warunki początkowe

```
#incjalizacja
xi = zeros((101, 3), float)
for i in range(0, 10): xi[i, 0] = 0.
for i in range(10, 20): xi[i, 0] = 0.05 * (i - 10) - 0.005
for i in range(21, 30): xi[i, 0] = -0.05 * (i - 20) + 0.015
for i in range(31, 70): xi[i, 0] = 0.
for i in range(71, 80): xi[i, 0] = 0.05 * (i - 70) - 0.035
for i in range(81, 90): xi[i, 0] = -0.05 * (i - 80) + 0.045
for i in range(91, 101): xi[i, 0] = 0.
for i in range(0, 101):
    string.append(vector(2.0 * i - 100.0, 300. * xi[i, 0], 0))
```

Rysunek 3: Inicjalizacja warunków początkowych

„Zeros” jest klasą biblioteki numpy, która tworzy macierz o wymiarach (101, 3), wypełnioną zerami. Klasa ta jest nam potrzebna do obliczania kroków czasowych równania.
Warunki początkowe pokazują także, że drgania struny zostały zadane w 2 punktach między wartościami 0.

3 Rozwiązanie poprzez rozszerzenie w trybie normalnym

Mody normalne systemu oscylacyjnego z jakim tutaj mamy do czynienia oznaczają ruch, w którym wszystkie części poruszają się sinusoidalnie z tą samą częstotliwością i ze stałą zależnością fazową.
Równanie (1) możemy rozwinąć i rozwiązać posługując się metoda rozdzielenia zmiennych.
Rozwiązaniem tego równania jest wynik działa funkcji czasu i przestrzeni.

$$y(x,t) = X(x)T(t)$$

Stosując tą metodę otrzymujemy rozwiązanie równania w postaci dwóch różniczek zwyczajnych.

$$\frac{d^2T(t)}{ddt^2} + \omega^2 T(t) = 0$$

$$\frac{d^X(x)}{dx^2} + k^2 X(x) = 0$$

$$k \overset{def}{=} \frac{\omega}{c}$$

Prędkość kątowa uzależniona jest od warunków brzegowych, które określają punkty zaczepienia struny – są jej węzłami.

$$X(x=0,t) = X(x=l,t) = 0$$

$$\Longrightarrow X_n(x) = A_n sink_n x, \quad k_n = \frac{\pi(n+1)}{L}, \quad n = 0,1,...$$

Rozwiązaniem dla składowej czasu jest natomiast:

$$T_n(t) = C_n sin\omega_n t + D_n cos\omega_n t, \quad \omega_n = nck_0 = n\frac{2\pi c}{L}$$

gdzie częstotliwość n-tego modu normalnego będzie stała. Oznacza to, że częstotliwość oscylatora, jest zdefiniowana przez mod normalny. Warunek początkowy (rys. 3) zerowej prędkości, wymaga aby wartość w Cn była równa zero. Razem to wszystko składa się na równanie modu normalnego.

$$y_n(x,t) = sink_n x cos\omega_n t, \quad n = 0,1,...$$

Jest to pojedyncze drganie oscylatora, dla fali możemy przedstawić to równanie w postaci sumy.

4 Algorytm kroku czasowego

Wyjściem dla tego algorytmu jest równanie Laplace'a, które dotyczyło dwóch zmiennych w przestrzeni. Dla naszego algorytmu będą to zmienne przestrzeni i czasu. Za pomocą równania Laplace'a dokonujemy dyskretyzacji równania falowego do różniczki. Zaczynamy od wyrażenia drugich pochodnych w postaci różniczki zupełnej.

$$\frac{\partial^2 y}{\partial t^2} \simeq \frac{y_{i,j+1} + y_{i,j-1} - 2y_{i,j}}{(\Delta t)^2}, \quad \frac{\partial^2 y}{\partial x^2} \simeq \frac{y_{i,j+1} + y_{i,j-1} - 2y_{i,j}}{(\Delta x)^2}$$

Po wstawieniu tych różniczek do równania falowego (1) otrzymujemy:

$$\frac{y_{i,j+1} + y_{i,j-1} - 2y_{i,j}}{c^2(\Delta t)^2} = \frac{y_{i,j+1} + y_{i,j-1} - 2y_{i,j}}{(\Delta x)^2}$$

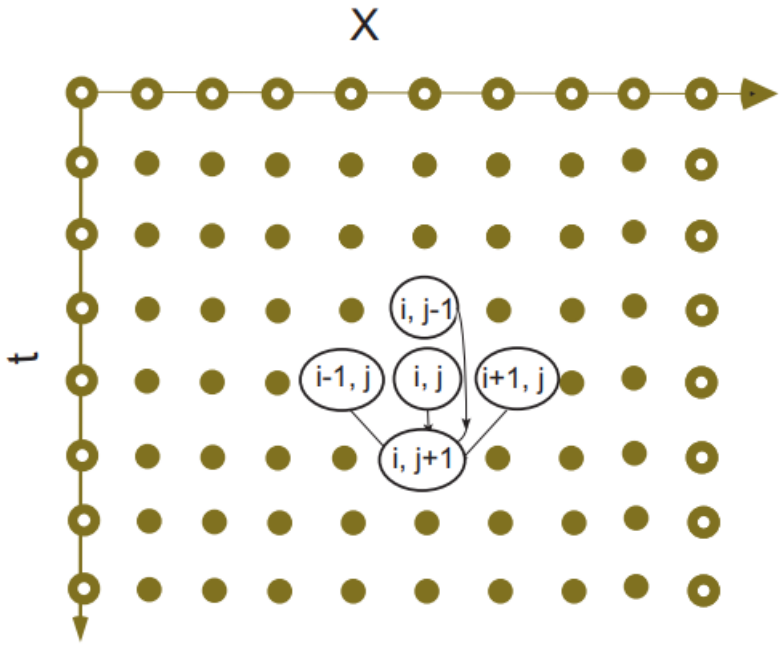
Równanie jak widać zawiera dwie składowe „i" odpowiadające składowej x-owej oraz j odpowiadające składowej czasu. Jak widać składowa czasu ma 3 wartości, które określamy jako:
przeszłość (j - 1), teraźniejszość (j) oraz przyszłość (j + 1).
Ostatnią składową czasu wyliczamy znając wartości dwóch poprzednich stanów zgodnie z równaniem:

$$y_{i,j+1} = 2y_{i,j} - y_{i,j-1} + \frac{c^2}{c'^2}[y_{i+1,j} + y_{i-1,j} - 2y_{i,j}], \quad c' \overset{def}{=} \frac{\Delta x}{\Delta t}$$

Przy czym c' jest kombinacją numerycznych parametrów przestrzeni i czasu.

Inicjalizacja algorytmu nie może nastąpić wprost ponieważ musimy znać przemieszczenie dla dwóch wartości czasu, a warunki początkowe dają tylko jedna wartość. Problem można rozwiązać ekstrapolując (central difference – środkowy wyraz dla rożniczek skończonych) z pozostałymi warunkami.

$$\frac{\partial y}{\partial t}(x,0) \simeq \frac{y(x,\Delta t) - y(x,-\Delta t)}{2\Delta t} = 0, \Longrightarrow y_{i,0} = y_{i,2}$$



Rysunek 4: Ilustracja kroku czasowego dla czasu i przestrzeni

```
#time step
for i in range(0, 100):
    xi[i, 1] = xi[i, 0] + 0.5 *ratio * (xi[i + 1, 0] - 2 * xi[i, 0])
while 1:
    rate(20)
    for  i in range(0, 100):
        xi[i, 2] = 2. * xi[i, 1] - xi[i, 0] + ratio * (xi[i+1, 1] + xi[i-1, 1] - 2 * xi[i, 1])
    for i in range(0, 101):
        x = 2. * i - 100
        y = 300. * (xi[i, 2])
        if x == -98 or x == 100: y = 0
        string.append(vector(x, y, 0))
    for i in range(0, 100):
        xi[i, 0] = xi[i, 1]
        xi[i, 1] = xi[i, 2]
```

Rysunek 5: Implementacja kroku czasowego

Implementacja kroku czasowego - pierwsza pętla określa pozycje każdego punktu struny w momencie wzbudzenia ruchu. W pętli while natomiast sprawdzamy jak wyglądają dwa poprzednie punkty, żeby określić jakie wychylenie uzyska punkt następny. Ostatnia pętla czyści macierz przez co wizualizacja może wykonywać się cały czas.

5 Wynik

fale.py

```
#! /usr/bin/python3

from vpython import *
from numpy import zeros

#elementy wizualizacji
scene = canvas(width=600, height=900, title='Vibrating string')
string = curve(vector(0, 0, 0), retain=100, color=color.yellow, radius=2)
ball1 = sphere(pos=vector(100, 0, 0), color=color.red, radius=2)
ball2 = sphere(pos=vector(-99, 0, 0), color=color.red, radius=2)

#Parametry
rho = 0.01
ten = 40
c = sqrt(ten/rho)
cl = c
ratio = c*c/(cl*cl)

#incjalizacja
xi = zeros((101, 3), float)
for i in range(0, 10): xi[i, 0] = 0.
for i in range(10, 20): xi[i, 0] = 0.05 * (i - 10) - 0.005
for i in range(21, 30): xi[i, 0] = -0.05 * (i - 20) + 0.015
for i in range(31, 70): xi[i, 0] = 0.
for i in range(71, 80): xi[i, 0] = 0.05 * (i - 70) - 0.035
for i in range(81, 90): xi[i, 0] = -0.05 * (i - 80) + 0.045
for i in range(91, 101): xi[i, 0] = 0.
for i in range(0, 101):
    string.append(vector(2.0 * i - 100.0, 300. * xi[i, 0], 0))

#time step
for i in range(0, 100):
    xi[i, 1] = xi[i, 0] + 0.5 *ratio * (xi[i + 1, 0] - 2 * xi[i, 0])
while 1:
    rate(20)
    for  i in range(0, 100):
        xi[i, 2] = 2. * xi[i, 1] - xi[i, 0] + ratio * (xi[i+1, 1] + xi[i-1, 1] - 2 * xi[i, 1])
    for i in range(0, 101):
        x = 2. * i - 100
        y = 300. * (xi[i, 2])
        if x == -98 or x == 100: y = 0
        string.append(vector(x, y, 0))
    for i in range(0, 100):
        xi[i, 0] = xi[i, 1]
        xi[i, 1] = xi[i, 2]
```

Rysunek 6: Program fale.py

6 Podsumowanie

Osiągnęliśmy cel naszego projektu opracowujac model propagacji fal na strunie oraz implementując program fale.py. Efekt działania naszego programu został zaprezentowany na filmiku załączonym do raportu.