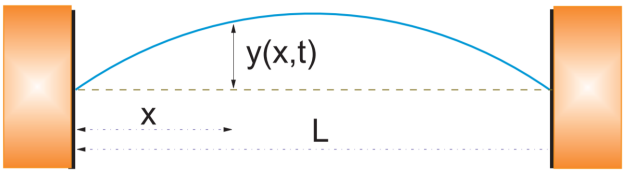
Modelowanie komputerowe

Raport – drgająca struna

Autorzy:
Budziło Maria,
Kubański Marcin,
Stypuła Hiacynta

11 stycznia 2022



[h!]

Rysunek 1: Struna z przywiązanymi końcami.

1. Wstęp

Tematem prezentacji jest drganie struny.

Pierwszą rzeczą, o której wspomniano był eksperyment z fizyki elementarnej. Następnie przedstawiono opis problemu, implementację programu dotyczącego drgania struny w języku Python wraz z rozwiązaniem problemu.

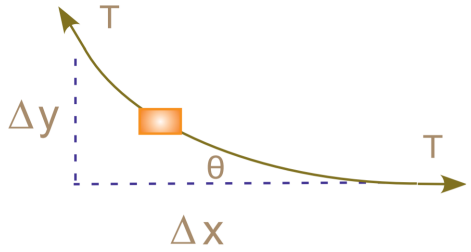
2. Opis problemu

Należy przypomnieć sobie demonstrację z fizyki elementarnej, w której obiektem badań jest sznurek o związanych obu końcach. Sznurek ten delikatnie jest szarpany w danym miejscu. W konsekwencji obserwuje się pulsowanie wzdłuż struny. Podobnie, jeśli jeden koniec struny byłby wolny i potrząśniętoby nim w odpowiedni sposób, to model zachowania się takiej fali stojącej będzie ustanowiony jako taki, w którym węzły pozostaną na miejscu, a anty węzły będą poruszały się po prostu w górę i w dół.

Tak też problemem z którym się zetknięto było opracowanie modelu propagacji fali na strunie.

3. Równanie hiperboliczne fali

Wzięto pod uwagę strunę o długości L, przywiązanej z dwóch końców. Struna ma stałą gęstość ρ na jednostkę długości, stałe naprężenie T i nie ma żadnych sił tarcia. Ustalamy, że naprężenie jest tak duże, że możemy zaniedbać oddziaływanie. grawitacyjne. Założono, że przemieszczenie struny y(x, t) od położenia spoczynkowego następuje tylko w kierunku pionowym i jest funkcją poziomego położenia wzdłuż struny x i czasu t.



Rysunek 2: Struna z przywiązanymi końcami.

Na powyższym rysunku jest widoczny element różniczkowy struny, który pokazuje, w jaki sposób przemieszczenie tej struny prowadzi do wywierania siły sprężystości, żeby przeciwdziałać sile naciągu.

Aby otrzymać proste liniowe równanie ruchu, zakłada się, że względne przemieszczenie struny $\frac{y(x,t)}{L}$ i nachylenie $\frac{\partial y}{\partial x}$ są małe. Wydzielony zostaje nieskończenie mały odcinek x struny i widać, że różnica pionowych składowych napięcia na obu końcach struny wytwarza siłę przywracającą, która przyspiesza ten odcinek struny w kierunku pionowym. Stosując prawa Newtona do tego odcinka, otrzymuje się znane równanie falowe:

$$\sum F_y = \rho \cdot \Delta x \cdot \frac{\partial^2 y}{\partial t^2}$$

$$= T \sin\theta(x + \Delta x) - T \sin\theta(x)$$

$$= T \frac{\partial y}{\partial x} \mid_{x+\Delta x} - \frac{\partial y}{\partial x} \mid \mid_x \approx T \frac{\partial^2 y}{\partial x^2}$$

$$\Rightarrow \frac{\partial^2 y(x,t)}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 y(x,t)}{\partial t^2}, c = \sqrt{\frac{T}{\rho}}$$

Założono, że θ jest wystarczająco małe, aby $\theta \simeq tan\theta = \frac{\partial y}{\partial x}$. Istnienie dwóch niezależnych zmiennych x i t czyni z tego równanie różniczkowe cząstkowe. Stała c jest prędkością, z jaką zaburzenie przemieszcza się wzdłuż fali i widać, że maleje ona dla cięższej struny, a rośnie dla węższej. Zauważyć można, że ta prędkość sygnału c jest różna od prędkości elementu struny $\frac{\partial y}{\partial t}$.
Warunki początkowe to delikatne szarpnięcie i zwolnienie struny. Zakładamy, że szarpnięcie powoduje ustawienie struny na kształt trójkąta ze środkiem trójkąta 8/10 w dół struny i o wysokości 1:

$$y(x, t = 0) = \begin{cases} 1.25x/L, & x 0.8L \\ (5 - 5x/L) & x > 0.8L \end{cases}$$

A ponieważ wcześniej omawiane przez nas otrzymane równanie fali jest równaniem różniczkowym drugiego rzędu, to potrzeba drugiego warunku początkowego, abyśmy mogli określić rozwiązanie. Interpretujemy pociągnięcie jak zwolnienie struny ze spoczynku:

$$\frac{\partial y}{\partial t}(x, t = 0) = 0$$

.

Natomiast warunki brzegowe powodują, że oba końce sznurka są przez cały czas związane:

$$y(0, t) \equiv 0,$$

$$y(L, t) \equiv 0.$$

4. Rozwiązanie numeryczne

Rozwiązanie analityczne omawianego równania fali, które otrzymaliśmy, otrzymuje się za pomocą znanej techniki rozdzielania zmiennych. Zakładamy, że rozwiązanie jest iloczynem funkcji położenia i funkcji czasu: $y(x, t) = X(x)T(t)$

Równanie powyższe podstawiliśmy do naszego równania fali i podzieliliśmy przez y(x,t). W ten sposób mamy równanie, które ma rozwiązanie tylko wtedy, jeżeli istnieją rozwiązania dla dwóch równań różniczkowych zwyczajnych:

$$\frac{d^2T(t)}{dt^2} + \omega^2 T(t) = 0, \quad \frac{d^2X(x)}{dx^2} + k^2 X(x) = 0, \quad k \stackrel{def}{=} \frac{\omega}{c}$$

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

Częstotliwość kątowa ω i wektor falowy k są określane przez warunek, aby rozwiązania spełniały warunki brzegowe. W szczególności wymaga tego struna przymocowana na obu końcach:

$$\Rightarrow X_n(x) = A_n \sin k_n x, \quad k_n = \frac{\pi(n+1)}{L}, \quad n = 0, 1, ...$$
$$T_n(t) = C_n \sin \omega_n t + D_n \cos \omega_n t, \quad \omega_n = nck_0 = n\frac{2\pi c}{L}$$

I dlatego otrzymujemy równanie tej postaci: $y_n(x, t) = \sin k_n x \cos \omega_n t, \quad n = 0, 1....$

Wtedy równanie czasowe wygląda w taki sposób. Gdzie częstotliwość n-tego modu normalnego jest także ustalona. W rzeczywistości będzie to pojedyncza częstotliwość oscylacji, która definiuje mod normalny. Natomiast warynek poczatkowy prędkości zerowego,

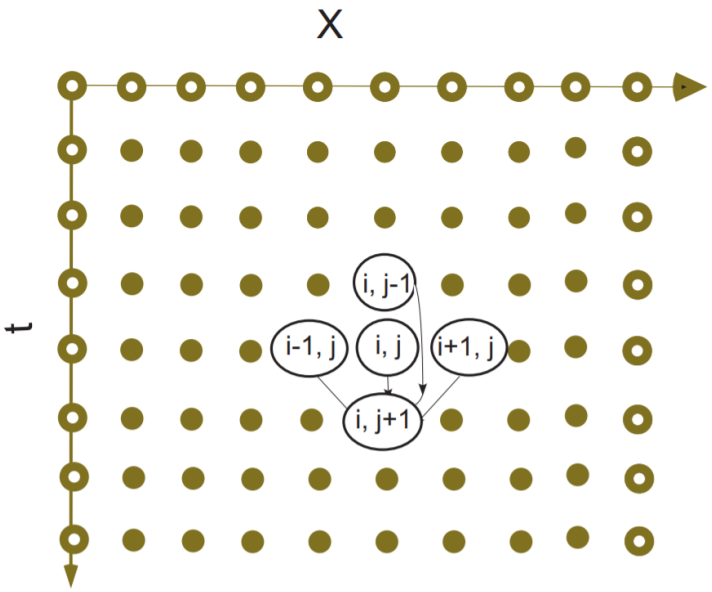
$\partial y/\partial t(t=0)=0$, wymaga, żeby wartości C_n były równe zero.
Ponieważ nasze równanie falowe jest liniowe w y, to obowiązuje zasada liniowej superpozycji, dlatego najbardziej ogólne rozwiązanie dla fal na strunie o stałych końcach można zapisać jako sumę modów normalnych Choć oczywiście tracimy liniową superpozycję jeżeli uwzględnimy nieliniowe człony w równaniu. $y(x,t) = \sum_{n=0}^{\infty} B_n \sin k_n x \cos \omega_n t$

$y(x, t=0) = \sum_{n}^{\infty} B_n \sin nk_0 x$

B_m to współczynnik Fouriea i jest on określony przez ostatni w kolejności z wymienionych przez nas warunek początkowy. Opisuje on w jaki sposób zrywana jest fala.

I tak też dostaliśmy przybliżone rozwiązanie numeryczne dla współczynnika Bm: $B_m = 6.25 \cdot \frac{\sin(0.8m\pi)}{m^2\pi^2}$. I oczywiście precyzja rozwiązania analitycznego jest zależna od liczby członów, które zsumujemy, a jednak liczba tych członów jest nieskończona, stąd rozwiązanie będzie zawsze przybliżone. Im więcej członów zsumujemy, tym bardziej precyzyjny wynik otrzymamy.

5. Algorytm kroku czasowego



Rysunek 3: Poszukiwanie rozwiązania $y(x,t)$ dla dyskretnych wartości zmiennych niezależnych x i t na siatce.

Podobnie jak w przypadku równania Laplace'a, szukane jest rozwiązanie $y(x,t)$ tylko dla dyskretnych wartości zmiennych niezależnych x i t na siatce, tak jak to przedstawia rysunek powyżej. W przeciwieństwie do równania Laplace'a, w którym siatka była w dwóch wymiarach przestrzennych, siatka jak na załączonym rysunku dotyczy zarówno przestrzeni, jak i czasu. Tutaj poruszanie się po rzędzie odpowiada wzrastającym wartościom x wzdłuż struny dla określonego czasu, podczas gdy poruszanie się w dół kolumny odpowiada zwiększaniu się kroków czasowych dla ustalonej pozycji. Nie można użyć techniki relaksacyjnej, ponieważ nie są znane rozwiązania na wszystkie cztery strony. Warunki brzegowe określają rozwiązanie po prawej i lewej stronie, podczas gdy początkowy warunek czasu określa rozwiązanie wzdłuż góry.

Podobnie jak w przypadku równania Laplace'a, do dyskretyzacji równania falowego w równanie różnicowe używa się przybliżenia z różnicą centralną. Najpierw wyrażona zostaje druga pochodna poprzez metodę różnic skończonych.

$$x = i\Delta x; i = 1, ..., N_x$$

$$t = j\Delta t; j = 1, ..., N_t$$

$$y(x,t) = y(i\Delta x, i\Delta t) \stackrel{def.}{=} y_{i,j}$$

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

A jeśli wstawić powyższe do równania falowego:

$$\frac{\partial^2 y(x,t)}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 y(x,t)}{\partial t^2},$$

gdzie $c = \sqrt{\frac{T}{\rho}}$, to otrzyma się: $\frac{y_{i,j+1}+y_{i,j-1}-2y_{i,j}}{(c^2\Delta t)^2} = \frac{y_{i+1,j}+y_{i-1,j}-2y_{i,j}}{(\Delta t)^2}$.

Po wstawieniu otrzymanego członu do wcześniej przedstawionego równania falowego $\frac{\partial^2 y(x,t)}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 y(x,t)}{\partial t^2}$, gdzie $c = \sqrt{\frac{T}{\rho}}$, to otrzymuje się taką zależność:

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

To równanie zawiera trzy wartości czasu: $j+1$, czyli przyszłość, j to teraźniejszość i $j-1$ to przeszłość. Dlatego też przedstawione zostanie to równanie w formie, która pozwoli przewidywać przyszłe rozwiązanie z rozwiązań przeszłych i rozwiązań teraźniejszych.

Stąd otrzymaliśmy takie równanie:

$$y_{i,j+1} = 2y_{i,j} - y_{i,j-1} + \frac{c^2}{c'^2}[y_{i+1,j} + y_{i-1,j} - 2y_{i,j}].$$

W tym równaniu c', $c' \stackrel{def.}{=} \frac{\Delta x}{\Delta t}$ jest kombinacją parametrów liczbowych z wymiarem prędkości, której rozmiar w stosunku do c określa stabilność algorytmu. Algorytm ten, tutaj przedstawiony, propaguje falę z dwóch wcześniejszych czasów, j i $j-1$ oraz z trzech pobliskich pozycji: $i-1$, i oraz $i+1$ do czasu późniejszego $j+1$ i pojedynczej pozycji w przestrzeni i.

Zapisanie zależności rekurencyjnej jest trudna, ponieważ wymaga przemieszczeń z dwóch wcześniejszych czasów, podczas gdy warunki początkowe są tylko dla jednego czasu. Jednak wykorzystując wcześniej poznane zależności na przedstawione jako pierwsze warunki początkowe oraz wykorzystując także aproksymację różnicy centralnej, można dokonać ekstrapolacji, czyli przewidywania wartości funkcji poza zakresem dla którego posiada się dane, poprzez dopasowanie do istniejących danych pewnej funkcji i następnie wyliczenie na tej podstawie jej wartości w szukanym przez nas punkcie, czyli tutaj do czasu ujemnego. Wtedy otrzyma się zależność:

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

Przyjęto czas poczatkowy jako $j=1$, więc $j=0$ odpowiada $t=-\Delta t$. I biorąc to pod uwagę, z zależności

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

otrzymano relację:

$$y_{i,2} = y_{i,1} + \frac{c^2}{2c'^2}[y_{i+1,1} + y_{i-1,1} - 2y_{i,1}]; j = 2.$$

To równanie wykorzystuje rozwiązanie w całej przestrzeni w czasie początkowym t=0 do propagacji do przodu do czasu Δt. Kolejne przedziały wykorzystują zależność

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

Powodzenie metody numerycznej zależy od względnych rozmiarów kroków czasu i przestrzeni. Warunek Courant'a oznacza, że rozwiązanie staje się lepsze, bardziej dokładne wraz z mniejszymi krokami czasowymi, ale pogarsza się dla mniejszych kroków przestrzennych, oczywiście chyba że jednocześnie zmniejszyliśmy krok czasowy to wtedy się nie pogorszy. To równanie falowe jest symetryczne w x i t, ale symetria jest łamana poprzez niesymetryczne warunki początkowe i warunki brzegowe:

$$c \leqslant c' = \Delta x/\Delta t \quad \text{Warunek Courant'a.}$$

6. Rozwiązanie problemu

```
# EqString.py:  Animated leapfrog solution of wave equation

from visual import *

# Set up curve
g = display(width = 600, height = 300, title = 'Vibrating string')
vibst = curve(x = list(range(0, 100)), color = color.yellow)
ball1 = sphere(pos = (100, 0), color = color.red, radius = 2)
ball2 = sphere(pos = ( - 100, 0), color = color.red, radius = 2)
ball1.pos
ball2.pos
vibst.radius = 1.0

# Parameters
rho    = 0.01                                          # string density
ten    = 40.                                           # string tension
c      = sqrt(ten/rho)                                # Propagation speed
cl     = c                                             # CFL criterium
ratio =  c*c/(cl*cl)

# Initialization
xi = zeros((101,3), float)                             # 101 x's & 3 t's
for i in range(0, 81):      xi[i, 0] = 0.00125*i;                 # IC
for i in range (81, 101):  xi[i, 0] = 0.1 - 0.005*(i - 80)        # IC
for i in range(0, 100):                                      # 1st t step
    vibst.x[i] = 2.0*i - 100.0                             # assign,scale x
    vibst.y[i] = 300.*xi[i, 0]                             # assign, scale y
vibst.pos                                                  # draw string

# Later time steps
for i in range(1, 100): xi[i,1] = xi[i,0] + 0.5*ratio*(xi[i+1,0]+xi[i-1,0]-2*xi[i,0])
while 1:                                  # continue plotting till user quits
    rate(50)                             # delays plotting, (bigger = slower)
    for i in range(1, 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(1, 100):
         vibst.x[i] = 2.*i - 100.0                                  # scaled x
         vibst.y[i] = 300.*xi[i, 2]                                 # scaled y
    vibst.pos                                                  # plot string
    for i in range(0, 101):
        xi[i, 0] = xi[i, 1]                                   # recycle array
        xi[i, 1] = xi[i, 2]

print("Done!")
```

Rysunek 4: Program wyjściowy stworzony w celu rozwiązywania równania falowego poprzez kroki czasowe dla struny o długości $L = 1m$ z nieruchomymi końcami.

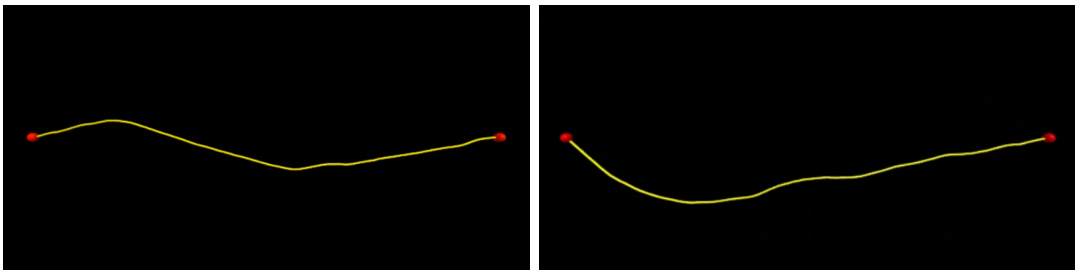
Powyższy obrazek przedstawia program wyjściowy mający na celu rozwiązywanie równania falowego poprzez kroki czasowe dla struny o długości $L = 1m$ z nieruchomymi końcami i z dobranymi warunkami początkowymi. Oczywiście kod ten poddano modyfikacjom, aby był on poprawny.

7. Rozwiązanie problemu

```
c = math.sqrt(ten / rho)  # 4000
c1 = c
steps = 101

g = canvas(width=600, height=300, title="Vibrating string")
string = curve(pos=[vec(i - steps / 2, 0, 0) for i in range(steps)], color=color.yellow, radius=0.5, x0=steps * [0.],
               x1=steps * [0.], x2=steps * [0.])
ball1 = sphere(pos=vector(100, 0, 0), radius=2.0, color=color.red)
ball2 = sphere(pos=vector(-100, 0, 0), radius=2.0, color=color.red)

for i in range(0, 81):
    string.x1[i] = (0.00125 * i + 0.5 * ratio * (0.00125 * (i + 1) + 0.00125 * (i - 1) - 2 * 0.00125 * i))
    string.x0[i] = string.x1[i]
    string.modify(i, y=300 * string.x2[i], x=2. * i - 100.)
for i in range(81, 101):
    string.x1[i] = (0.1 - 0.005 * (i - 80) + 0.5 * ratio * (
                0.1 - 0.005 * (i - 80 + 1) + 0.1 - 0.005 * (i - 80 - 1) - 2 * 0.1 - 0.005
                * (i - 80)))
    string.x0[i] = string.x1[i]
    string.modify(i, y=300 * string.x2[i], x=2. * i - 100.)


nstep = 0
while 1:
    rate(500)

    for i in range(1, 100):
        string.x2[i] = 2. * string.x1[i] - string.x0[i] + ratio * (
                string.x1[i + 1] + string.x1[i - 1] - 2 * string.x1[i])
        string.modify(i, y=300 * string.x2[i], x=2. * i - 100.)

    for i in range(0, 101):
        string.x0[i] = string.x1[i]
        string.x1[i] = string.x2[i]
    nstep += 1
    if nstep == 1: g.waitfor('click')

```

Rysunek 5: Napisany program rozwiązujący problem drgającej struny.

8. Implementacja. Podsumowanie



Rysunek 6: Implementacja kodu rozwiązującego problem drgającej struny.

9. Bibliografia

Materiały udostępnione w pliku „2Fale.pdf”
`https://www.vpython.org/`