Simulating 1D Waves in Python

Sorn Zupanic Maksumic

February 2021

Contents

1 Overview

This document shows how to solve the 1D wave equation (Equation 1) in Python. The solution is approximated with the central finite difference method (also known as the Crank-Nicholson scheme). The algorithm description is influenced by [1] and [2].

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

1.1 Terminology

It's called a 1D wave equation even though $u(x,t)$ has two inputs: x and t. This is because we only count the number of inputs in the spacial domain.

The output $u(x,t)$ is the y-coordinate in space, for the given x and t.

1.2 Dependencies

The Python script depends on the following 3$^{\text{rd}}$ party libraries:

- Matplotlib
- Numpy

1.3 Result

The program outputs an animated GIF file. The animation shows six 1D waves with different velocities.

2 Algorithm Description

This document uses u_i^n as a concise notation for $u(x_i, t_n)$. Similarly, u_{ij}^n is a concise notation for $u(x_i, y_j, t_n)$.

2.1 Initial Conditions

The Initial Condition (IC) is shown in Equation 2. It returns u for all x_i when $t = 0$. In other words, it returns the start position for each x_i. The equation set given in Equation 2 produces a "triangle-shaped" IC.

$$I(x_i) = u_i^0 = u(x_i, t_0) = \begin{cases} u(i\Delta x, 0) = \frac{A}{\bar{x}}x, 0 \leq x \leq 1 \\ u(i\Delta x, 0) = \frac{A}{L-\bar{x}}(L-x), 1 \leq x < L \end{cases} \tag{2}$$

The IC for the first-order derivative is zero (see Equation 3). In other words, there is no initial velocity. This is important in a later step, because it lets us approximate Equation 3 with a first-order partial central difference and get Equation 4.

$$\frac{\partial}{\partial t}u(x, 0) = 0 \tag{3}$$

$$\frac{\partial}{\partial t}u(x_i, t_0) \approx \frac{u_i^1 - u_i^{-1}}{2\Delta t} = 0 \implies u_i^1 = u_i^{-1} \tag{4}$$

2.2 Boundary Conditions

The boundary conditions are shown in Equation 5, where L is the length of the wave. In other words, the wave is *fixed* at both ends.

$$u(0, t) = 0 \ \wedge \ u(L, t) = 0 \tag{5}$$

2.3 Central Finite Difference

Start with the 1D wave equation (Equation 1) and replace both sides of the equation with second-order central finite differences. The second-order central finite differences are given as Equation 6 and 7.

$$\frac{\partial^2}{\partial t^2}u(x_i, t_n) \approx \frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} \tag{6}$$

$$\frac{\partial^2}{\partial x^2}u(x_i, t_n) \approx \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \tag{7}$$

Result:

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} = c^2 \left(\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \right) \tag{8}$$

Solve for u_i^{n+1}:

$$u_i^{n+1} = C^2(u_{i+1}^n - 2u_i^n + u_{i-1}^n) + 2u_i^n - u_i^{n-1} \tag{9}$$

Here, C is Courant's number:

$$C = c\frac{\Delta x}{\Delta t} \tag{10}$$

The "problem" with Equation 9 is that we do not have u_i^{n-1}. Therefore, we use Equation 4 to substitute u_i^{n-1} with u_i^{n+1}, do some simple algebra, and get Equation 11:

$$u_i^{n+1} = \frac{1}{2}C^2 \left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right) + u_i^n \tag{11}$$

We use the IC to get all u_i^0. Then, solve u_i^1 with Equation 11. Now, that we have u_i^{n-1}, use Equation 9 to solve the rest.

That's it!

3 Code Explanation

Here, some parts of the code are explained. The code is commented with further explanations.

3.1 Solver

The first loop applies Equation 2 to find u_i^0 for all i:

```
for i in range(0, Nx+1):
    u[0][i] = I(i*dx, A, s, L)
```

Listing 1: Initial Condition

The second loop applies Equation 4 to find u_i^1 for all i:

```
for i in range(1, Nx):
    u[1][i] = 0.5*(C**2)*(u[0][i+1]-2*u[0][i]+u[0][i-1]) + \
              u[0][i]
```

Listing 2: Initial Condition

The third loop applies Equation 9 to solve the rest of the 1D wave equation.

```
for n in range(1, Nt):
    for i in range(1, Nx):
        u[n+1][i] = (C**2)*(u[n][i+1]-2*u[n][i]+u[n][i-1]) + \
                    2*u[n][i]-u[n-1][i]
```

Listing 3: Initial Condition

3.2 Animation

The reason for first looping over all the time steps and *then* over all the 1D waves, is to group the correct animations together. If you do it the other way around, it only display one wave at a time: first displaying the entire animation for the first 1D wave, then the second, ...

```
for n in range(T):
    plts_tmp = []
    for i in range(len(li)):
        u, x, color = svl[i][0], svl[i][1], svl[i][2]
        p = plt.plot(x, u[n][:] + offset * i, color)
        plts_tmp.append(*p)
    plts.append(plts_tmp)
```

Listing 4: Initial Condition

But by grouping the correct animations together it displays the animations for all the 1D waves at the same time.

References

[1] Hans Petter Langtangen. *Finite Difference Computing with Exponential Decay Models*. URL: https://hplgit.github.io/decay-book/doc/pub/book/pdf/decay-book-4print.pdf.

[2] Hans Petter Langtangen. *Finite difference methods for wave equations*. URL: https://hplgit.github.io/fdm-book/doc/pub/wave/pdf/wave-4print.pdf.