Report of the computational solution for 1D acoustic wave equation

Leevi Tuikka, 014583623

January 20, 2020

1 Introduction

2 Theory

2.1 Physical and mathematical foundations

The whole basis of this project lies on one-dimensional scalar wave equation which is second-order partial differential equation. Physically it describes, how wave, i.e. disturbance, advances in a physical medium (string in this case) as a function of time and space. One-dimensional wave equation looks the following

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

where $\frac{\partial^2}{\partial t^2}$ is second-order partial time derivative, $\frac{\partial^2}{\partial x^2}$ is second-order spatial derivative, $u(x)$ is the disturbance, x is the only spatial coordinate of the medium and c is the propagation speed of the disturbance, defined by the mechanical properties of the medium.

"Disturbance $u\left(x\right)$" might sound a bit abstract as it is, so let us limit wave type to only mechanical waves and particularly longitudinal waves. This wave type advances to same direction as the disturbance is. Therefore, in order to describe disturbance mathematically, no other variables (e.g. angles) than magnitude of the disturbance are needed, so we are really dealing with scalar object. We can also conclude, that as we are dealing with mechanical longitudinal waves producing scalar disturbances, those disturbances manifest as density changes in a continuous medium. Now, by cutting out possible temperature changes and changes of chemical composition of a medium, and looking at changes in mechanical properties leading to density changes, it becomes evident that "disturbance" is actually variation in mechanical pressure. Density variations also leads us to deformation, which poses elastic behaviour in this particular case, if we consider a medium to dense gas or elastic geological material. To relate previous discussion to equation (1), let us define square of the propagation speed

$$c^2 = \frac{K}{\rho} \tag{2}$$

where K is the bulk modulus, which defines how much a medium deforms as a function of stress and ρ is the density. For the sake of clarity, let us define $p\left(x,t\right) \equiv u\left(x,t\right)$, so it matches SI symbol.

Now, as we know what eq. (1) more or less presents, it makes sense to start putting it to some more useful form. Currently, eq. (1) assumes, that disturbance travels somewhere outside of the boundaries of the medium, which needs to be fixed for this project. Therefore we introduce the source term

$$s = f\left(x,t\right) \tag{3}$$

where $f(x,t)$ is some arbitrary function, as a function of place and time. We consider point source, i.e. the source function can "produce non-zero values" only in one spatial point, so source time function can be decomposed into Dirac

delta function $\delta(x)$ and some arbitrary function $g(t)$ which varies as a function of time

$$s(x,t) = \delta(x)\, g(t) \tag{4}$$

For this project, we use particular form[1] of the 1st derivative of Gaussian function for $g(x)$, so source function becomes the following

$$s(x,t) = \delta(x)\, g(t) = \delta(x) \cdot \left(-8 f_0 (t - t_0)\, e^{-16 f_0^2 (t-t_0)}\right) \tag{5}$$

As we have now defined the source function, complete form of the 1D scalar wave equation with source looks the following

$$\frac{\partial^2 p(x)}{\partial t^2} = c^2 \frac{\partial^2 p(x)}{\partial x^2} + s(x,t) \tag{6}$$

2.2 Discretization

To be able to transform eq. (6) into program code, it must be transformed from the continuous domain to the discretized domain. For this, let us define some arbitrary discretization point $x_j = j\mathrm{d}x$, where $j \in [0, j_{\max}]$ is the index of the particular discretization point and $\mathrm{d}x$ is the grid spacing coefficient, i.e. the distance between two discretization points, which is constant in this project. Also time domain must be discretized, so let us define some arbitrary time step as $t_n = n\mathrm{d}t$, where $n \in [0, n_{\max}]$ is the index of the time step and $\mathrm{d}t$ is the length of the time step. Using the previous definitions, pressure at a given place and time can be written

$$p(x_j, t_n) = p_j^n \tag{7}$$

when eq. (6) becomes with eq. (4)

$$\frac{\partial^2 p_j^n}{\partial t^2} = c_j^2 \frac{\partial^2 p_j^n}{\partial x^2} + \delta_j g^n \tag{8}$$

as we demand, that also propagation speed c may vary as a function of space. However, derivatives are still presented in a continuous fashion. By using finite difference method, first-order derivative is defined in the following way

$$\frac{\partial f(x)}{\partial x} = \lim_{dx \to 0} \frac{f(x + \mathrm{d}x) - f(x - \mathrm{d}x)}{\mathrm{d}x} \tag{9}$$

In order to deal with the $\lim_{dx \to 0}$, eq. (9) can be split to backward-difference scheme, central-difference scheme and forward-difference scheme

$$\begin{aligned}
\frac{\partial f^-(x)}{\partial x} &\approx \frac{f(x) - f(x - \mathrm{d}x)}{\mathrm{d}x} \\
\frac{\partial f^c(x)}{\partial x} &\approx \frac{f(x + \mathrm{d}x) - f(x - \mathrm{d}x)}{2\mathrm{d}x} \\
\frac{\partial f^+(x)}{\partial x} &\approx \frac{f(x + \mathrm{d}x) - f(x)}{\mathrm{d}x}
\end{aligned}$$

To pick the best one of these, we can estimate errors by using Taylor series expansion

$$f(x+\mathrm{d}x) \approx f(x) + f'(x)\,\mathrm{d}x + \frac{1}{2}f''(x)\,\mathrm{d}x^2 + O\left(\mathrm{d}x^3\right) \tag{10}$$

Backward-difference scheme and forward-difference scheme should be symmetrical for some symmetric function, so we can estimate error of both by picking either one and expanding it to Taylor series

$$\frac{\partial f^-(x)}{\partial x} \approx f'(x) + O(\mathrm{d}x)$$

And Taylor series expansion for the central-difference scheme

$$\frac{\partial f^c(x)}{\partial x} \approx f'(x) + O\left(\mathrm{d}x^2\right)$$

so it is evident that central-difference scheme converges fastest, i.e. gives best solution as the $\mathrm{d}x \to 0$.

Now, to construct discretized form for the 2nd order derivative, we would like to use of course central-difference scheme. This can be done by using backward-difference and forward-difference schemes of once differentiated function, since we are trying to find the slope of the slope. Now, the central-difference scheme for the second order derivative would be

$$\frac{\partial^2 f^c(x)}{\partial x^2} \approx \frac{\frac{\partial f^+(x)}{\partial x} - \frac{\partial f^-(x)}{\partial x}}{\mathrm{d}x} = \frac{f(x+\mathrm{d}x) - 2f(x) + f(x-\mathrm{d}x)}{\mathrm{d}x^2}$$

And plugging this to eq. (6) we get

$$\frac{p_j^{n+1} - 2p_j^n + p_j^{n-1}}{\mathrm{d}t^2} = c_j^2\left(\frac{p_{j+1}^n - 2p_j^n + p_{j-1}^n}{\mathrm{d}x^2}\right) + \delta_j g^n \tag{11}$$

In order to find p_j^{n+1}, i.e. value for the pressure on the next time step at some spatial point, eq. (11) must be rearranged to form

$$p_j^{n+1} = c_j^2\frac{\mathrm{d}t^2}{\mathrm{d}x^2}\left(p_{j+1}^n - 2p_j^n + p_{j-1}^n\right) + 2p_j^n - p_j^{n-1} + \mathrm{d}t^2\delta_j g^n \tag{12}$$

3 Program code and technical details

3.1 Pseudocode

```
import libraries;

input_file = open("input_file.txt");

# read input parameters to Input class structure;
input_values = read_file(input_file);

# create Source class object
source = create_source(input_values.get_source_id());

# init discretization point, propagation speed, density and bulk modulus arrays
points = [];
c = [];
density = [];
K = [];

# set right values of c, density and K for each point
c = input_values.get_c();
density = input_values.get_density();
K = input_values.get_K();

# add Point class objects to points array and set values for them
for i = 0, i < input_values.get_n_points(), i++;
    points.append(add_point(i, K[i], density[i], c[i], p=0.0, input_values));
    # allocate array for each point to store p values over time steps
    points[i].allocate_t_array(input_values.get_timesteps());

# start looping over time steps;
for t = 0, t < input_values.get_timesteps(), t++;
    # loop over points at every time step, expect at boundaries
    # since points on boundaries don't have j+1 or j-1 points
    for j = 1, j < input_values.get_n_steps() - 1, j++;
        # calculate value for source if in the source point
        if j == source.get_id();
            s = source.calculate(t, dt);
        else;
            s = 0;
        # calculate p^{n+1} for each point
        points[i].set_p_t(c[n]**2 * (input_values_get_dt()**2 / input_values_get_dt()**2)
        *(points[i+1].get_p_t(t) - 2*points[i].get_p_t(t) + points[i-1].get_p_t(t))
        + 2*points[i].get_p_t(t) - points[i].get_p_t(t-1) + input_values.get_dt()**2*s)
```

4 Results

5 References