Report of the computational solution for 1D acoustic wave equation

Leevi Tuikka, 014583623

January 21, 2020

1 Introduction

This project implements the computational solution for 1D acoustic wave equation. Acoustic wave equation is the basic equation in seismology, as we consider Earth's crust and upper mantle to deform elastically. In this report, the theory behind the program is handled first, then technical details of the program and results from the program as the last part.

In the file structure, video and image files can be found under `media/` and the source code and the input file under `src/`.

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

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

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

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

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

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

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

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

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

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

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

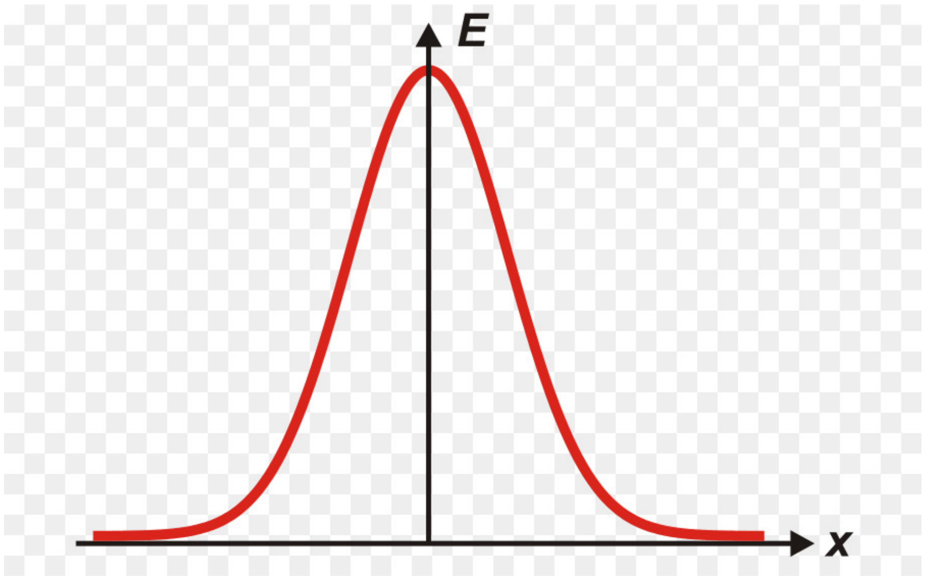
$$p_j^{n+1} = c_j^2\frac{\mathrm{d}t^2}{\mathrm{d}x^2}\left(p_{j+1}^n - 2p_j^n + p_{j-1}^n\right) + 2p_j^n - p_j^{n-1} + \mathrm{d}t^2\delta_j g^n \tag{12}$$

2.3 Source

Mathematically the source function eq. (5) is integrated over time. Therefore the waveform should Gaussian function

$$-8f_0 \int (t - t_0)\, e^{-16f_0^2(t-t_0)^2} = \frac{e^{-16f_0^2(t-t_0)^2}}{4f_0} \tag{13}$$

which looks graphically the following



3 Program code and technical details

3.1 Python

The program code was developed by using 3.6 version of Python, however it should have backward and forward compatibility to all Python 3.x versions. Implementation employs `numpy`, `matplotlib`, `math` libraries which should be included in Python standard libraries.

3.2 Instructions

Main functionality of the code is located in `main.py` file and class definitions to `class_defs.py` file. In order to run the program, run the following line on command line

```
$ python3.6 main.py
```

and the program first prints the input values to the terminal and after execution it opens a new window in order to plot results on $x - t$ domain. Quality of the image may vary but the program also saves the plot to file `x_t_plot.png` which should show the plot as intended.

3.3 Pseudocode

```
begin();

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(source_id, frequency);

# 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);

# plot p(t) values from point array over x-t domain
plot(points[i]);

end();
```

3.4 Classes

Implementation was done using Python classes for several functionalities. These include

- **Input**: This class structure handles input parameters. First, Input object is initialized by using input parameter variables. Therefore setter functions are not needed, since all the variables are set in the initialization stage. Propagation speed `c` is calculated in the initialization based on the bulk modulus `K` and the density `density`. Only other functionalities in the Input class are getter functions, which are used at any point where input parameters need to be used.

- **Point**: This class deals with single discretization points. Each point has properties: the spatial index `id`, the bulk modulus `K`, the density `density`, the propagation speed `c` and pressure array `p_t[]` over time steps. Point class structure contains setter and getter functions for each variable. It also contains `solve_p_t()` function which solves p_j^{n+1} based on the eq. (12). For debugging purposes, `print_point()` function exists, it prints properties of a certain Point object when called. In the beginning of the program, all the Point objects are initialized and appended to a array.

Later on in the time step loop, each point is accessed at every time step and p_j^{n+1} is computed by using class functionalities.

- **Source**: This class controls the source behaviour. The only variable that Source class object has, is spatial index `source_id` defining the source location. Source object is initialized in the beginning of the program by using `source_id`. Then it is accessed at every time step by Point objects, which check if current discretization point is the location of the source. If so, then value of the source function is computed as a function of time and the frequency by using `s_t()` function and then passed to `solve_p_t()` function.

4 Results

4.1 Parameters and different scenarios

Program implementation allows to use 1-3 different regions in the simulation, i.e. three different materials. Placement and width of the regions are controlled by the `regions` parameter which defines number of the regions, whereas the `interface1_id` defines interface between material 1 and material 2, and the `interface2_id` defines interface between material 2 and material 3, assuming that three regions are used. Accordingly, if two are used, only `interface1_id` is needed and in the case of one regions, no interface is needed.

All the input parameters are defined in the input file. By default, the material 1 has properties of air at 1 atm, the material 2 has bulk modulus of the material 1 and the density two times of material 1, the material 3 has bulk modulus of the material 1 and the density twenty times of material 1. Parameter `t_steps` defines the number time steps and `n_steps` defines number of discretization points. Product of these two parameters should be somewhere between $9 \cdot 10^6 - 25 \cdot 10^6$ for convenient run times and file sizes for output files.

Picking suitable values for the time step `dt` and the grid spacing `dx` was rather important and challenging part. If `dt` is too large, program can't catch all the smallest time-wise developments, e.g. regarding the source. Let us have the frequency of the source `freq` and time step `dt`, so period of source oscillation is `T`$= 1/$`freq` and program is able to catch `T/dt` "data points" from the source. For the source frequency, I picked 20 Hz (which roughly gives the biggest amplitude for eq. (5)) and I thought that 200-300 data points from the source should be enough, so `dt`$= 0.2 \cdot 10^{-3}$ seconds is suitable time step, since it gives 250 data points from the source. Selected `dt` also works fine in the respect of other things, as long as `dx` is suitable. Parameter `dx` can't be too large or otherwise the computational x domain can't transmit wave having relatively short wavelength. Now, considering material 1, wavelength in it is $\lambda = 343$ ms^{-1}/20 Hz = 17.15 meters, so to distribute one period over roughly 100 discretization points, I picked `dx`=0.1 meters. Also the ratio $($`dt/dx`$)^2$ must be big enough. For some reason, ratios less than 10^{-7} seems to lead weird disturbances in the waveform. Potentially this happens because last three terms in the eq. (12) become too dominant. The source function in the eq. (5) contains also time offset parameter `t0`, which defines when the source starts vibrating. Just by experimenting I picked `t0`=0.04 seconds, smaller values caused disturbances.

4.2 Reflections and refractions

When a wave hits an interface separating two different materials, some portion of it reflects back to its incident direction and rest of it refracts to a new material. These portions are defined by the incident angle and mechanical properties of the two media. However, as we are dealing with one-dimensional case, the incident angle is always 90°. Therefore, only mechanical properties, or to be exact, only propagation velocities matter. Reflection coefficient[2] defines how large portion of the amplitude (or actually energy) will be reflected to incident direction

$$R = \frac{c_2 - c_1}{c_2 + c_1} \tag{14}$$

where c_2 is the propagation velocity in the second medium and c_1 is the propagation velocity in the first medium. On the boundaries, we have set $p_j^{n+1} = 0$, so $c_2 = 0$ on the boundary points and outside of the computational domain which leads to $R = -c_1/c_1 = -1$. Physically this means, that the whole wave is reflected back but the amplitude is flipped, i.e. if wave having amplitude A hits the boundary, it will be reflected back having amplitude $-A$ and vice versa.

4.3 Quantifying the propagation velocity

In order to confirm that the program actually works, we can compare propagation velocity that has been given as a parameter, and propagation velocity which is quantified graphically from the output plot. Since the velocity is generally

$$v = \frac{\mathrm{d}x}{\mathrm{d}t} \tag{15}$$

but we can't use continuous derivatives here, let us define the propagation speed in the following way

$$c = \frac{x(t_2) - x(t_1)}{t_2 - t_1} \tag{16}$$

where $x(t_2)$ is the location of the crest at time t_2 and $x(t_1)$ is the location of crest at time t_1.

4.4 Example cases

4.4.1 A homogeneous medium

The first case was run only by using one regions, i.e. one material. For the number of discretization points I used 2000 points and 4000 time steps. Source was located in the middle, $x_j = 1000$. As an output the following $x - t$ plot was obtained.

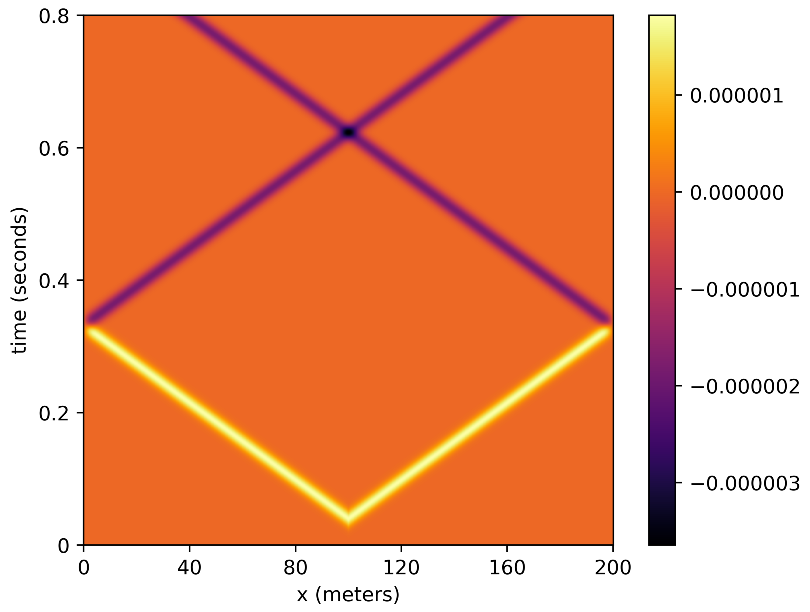
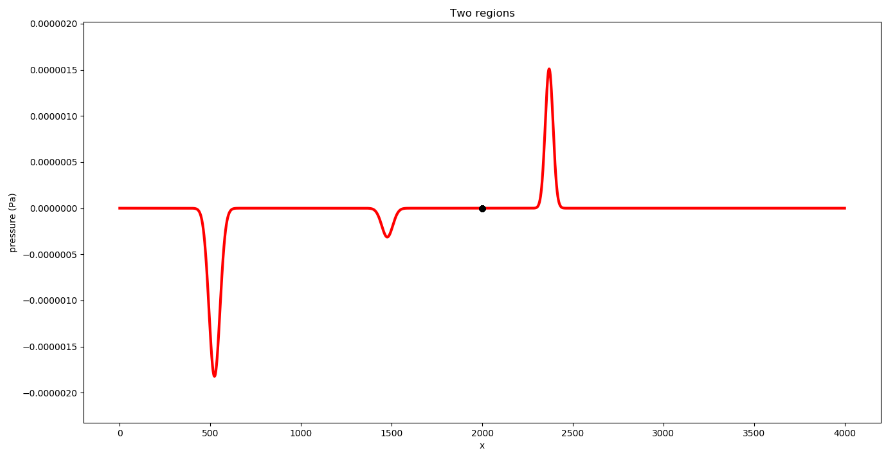


Figure 1: Colourbar presents the range of the pressure variation in pascals. Black dot at 0.6 seconds is the superposition of the two waves, i.e. the amplitude is $-2A$.

From this plot we are able estimate propagation speed. Let us pick roughly $t_1 = 0.04$ s, so $x(t_1) = 100$ m. Wave reaches boundaries roughly at $t_2 = 0.34$ s and $x(t_2) = 0$ m. This gives propagation speed $c = 100$ m/$(0.34 - 0.04)$ m $=$ 333.333... m/s which is quite close to the exact value 343 m/s, so I would conclude that the program works as it should. Also, at the boundaries, wave is flipped upside down, which is expected as well. Behaviour of the wave can be seen on video file `animation_1_region.mp4` and the high-resolution plot is in file `plot_1_region.png`.

4.4.2 Two regions

The second case was run by using two regions, i.e. two materials. For the number of discretization points I used 4000 points and 4000 time steps. Source was located at $x_j = 1000$ and the interface at $x_j = 2000$. In this case, the wave is reflected at the interface. The following is snapshot of the video file `animation_2_regions.mp4`, right away after the reflection (higher resolution image can be found `media/reflection.png`)



As the medium on the left of the interface has propagation speed $c_1 = 343$ m/s and the medium on the right of the interface has propagation speed $c_2 = \sqrt{141200 \text{ Pa}/2.4004 \text{ kg/m}^3} = 242.536...$ m/s. Now, by employing eq. (13), let us compute the reflection coefficient

$$R = \frac{242.536 \text{ m/s} - 343 \text{ m/s}}{242.536 \text{ m/s} + 343 \text{ m/s}} = -0.172... \tag{17}$$

i.e. 17.2% of the amplitude should be reflected back. By estimating amount of reflection graphically, the undisturbed amplitude is $1.8 \cdot 10^{-6}$ Pa, whereas the reflected the reflected amplitude is roughly $0.3 \cdot 10^{-6}$ Pa, so graphically estimating $0.3/1.8 = 16.666...\%$ of the amplitude was reflected, which is again quite a good result. Also the $x{-}t$ domain plot (high-resolution version `media/plot_2_regions.png`) looks the following

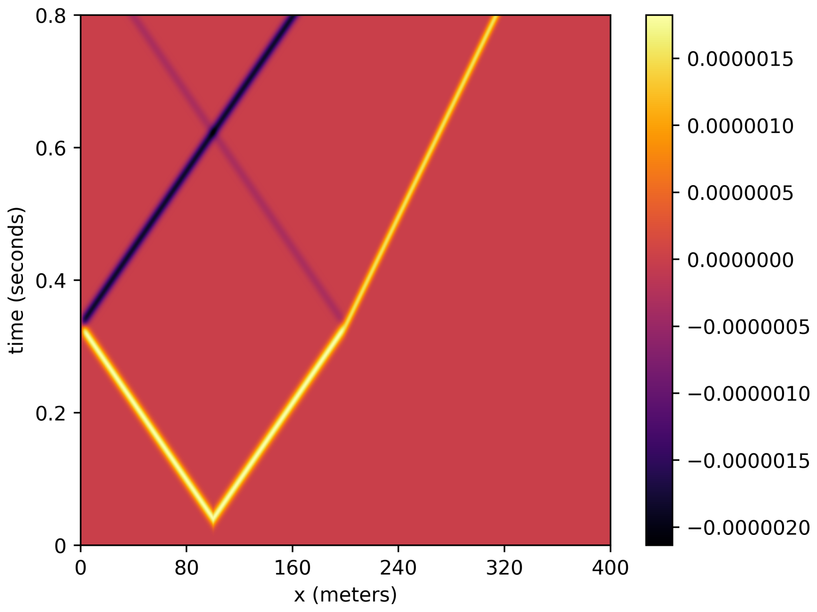


Figure 2: Colourbar presents the range of the pressure variation in pascals.

4.4.3 Three regions

The third case was run by using three regions, i.e. three materials. For the number of discretization points I used 4000 points and 4000 time steps. Source was located at $x_j = 1500$, the interface 1 at $x_j = 2000$ and the interface 2 at $x_j = 3000$. This simulation doesn't pose any useful example but rather demonstrates chaotic nature of the wave travelling in multiple media, which can been seen also on video `animation_3_regions.mp4`. The $x - t$ domain plot (high-resolution version `media/plot_3_regions.png`) looks the following

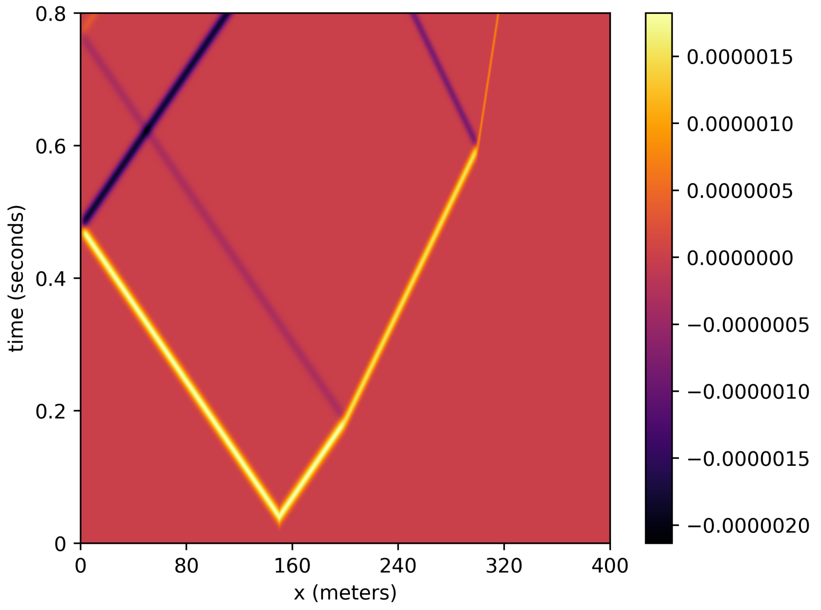


Figure 3: Colourbar presents the range of the pressure variation in pascals.

This however demonstrates quite well, how the wavelength is defined by the density, as the wavelength decreases drastically when wave passes from density 2.4004 kg/ms^2 to 24.004 kg/ms^2 at $x_j = 3000$.

5 References

[1] Igel, H. (2016) Computational Seismology: A Practical Introduction. 1st edn. Oxford: Oxford University Press. doi: 10.1093/acprof:oso/9780198717409.001.0001.

[2] Shearer, P. M. (2009) Introduction to Seismology. 2nd edn. Cambridge: Cambridge University Press. doi: 10.1017/CBO9780511841552.