Table of Contents

Computational Acoustics

- Ravi Kiran Chilakapati (chilar@uw.edu)

2nd Term paper for PHYS 536

The goal of this notebook is to computationally model acoustic and elastic wave propagation in homogeneous and isotropically non-homogeneous materials using the Finite difference method.

Preliminaries

In this section, we'll laydown some notation that will be used in the rest of the notebook.

Acoustic wave equation

The equation of motion of a forced/driven oscillator in gasses/liquids is usually expressed as

$$\partial_t^2 u(\mathbf{x}, t) = c(\mathbf{x})^2 \cdot \partial_{\mathbf{x}}^2 u(\mathbf{x}, t) + s(\mathbf{x}, t)$$

where, $u(\mathbf{x}, t)$ is the displacement (or pressure), $c(\mathbf{x})$ is the phase speed of the wave, $s(\mathbf{x}, t)$ is the externally applied force, $\mathbf{x}$ is the position vector (1D, 2D or 3D depending on context), and ∂_t and ∂_x are partial derivatives w.r.t. time and position respectively.

Elastic wave equation

The equation of motion of a forced/driven oscillator in an elastic medium (solids) is usually expressed as

$$\boldsymbol{\rho} \cdot \partial_t^2 u(\mathbf{x}, t) = \partial_{\mathbf{x}} \left(\mu(\mathbf{x}) \partial_{\mathbf{x}} u(\mathbf{x}, t) \right) + s(\mathbf{x}, t)$$

where $\boldsymbol{\rho}$ is the mass density of the medium, $u(\mathbf{x}, t)$ is the displacement caused by the wave, $\mu(\mathbf{x})$ is the shear modulus of the medium, $s(\mathbf{x}, t)$ is the externally applied force, $\mathbf{x}$ is the position vector (1D, 2D, or 3D depending on context), and ∂_t and ∂_x are partial derivatives w.r.t. time and position respectively.

If we assume a homogeneous medium with a constant shear modulus μ and constant mass density ρ, the elastic wave equation can be cajoled into the form of the pressure wave equation described above, with $c = \sqrt{\frac{\mu}{\rho}}$

Finite Difference Method

Preliminaries

Numerical Derivatives

The forward derivative is defined as

$$\frac{\mathrm{d}f}{\mathrm{d}x}^{forward} = \lim_{\mathrm{d}x \to 0} \frac{f(x+\mathrm{d}x) - f(x)}{\mathrm{d}x}$$

The centered derivative is defined as

$$\frac{\mathrm{d}f}{\mathrm{d}x}^{centered} = \lim_{\mathrm{d}x \to 0} \frac{f(x+\mathrm{d}x) - f(x-\mathrm{d}x)}{2\,\mathrm{d}x}$$

The backward derivative is defined as

$$\frac{\mathrm{d}f}{\mathrm{d}x}^{backward} = \lim_{\mathrm{d}x \to 0} \frac{f(x) - f(x-\mathrm{d}x)}{\mathrm{d}x}$$

These are exact definitions.

Numerically, the derivatives can be approximated (when $\mathrm{d}x$ is small, but not 0) as below:

$$\begin{aligned}
\frac{\mathrm{d}f}{\mathrm{d}x}^{forward} &\approx \frac{f(x+\mathrm{d}x) - f(x)}{\mathrm{d}x} \\
\frac{\mathrm{d}f}{\mathrm{d}x}^{centered} &\approx \frac{f(x+\mathrm{d}x) - f(x-\mathrm{d}x)}{2\,\mathrm{d}x} \\
\frac{\mathrm{d}f}{\mathrm{d}x}^{backward} &\approx \frac{f(x) - f(x-\mathrm{d}x)}{\mathrm{d}x}
\end{aligned}$$

Derivatives computed using these approximations are called Finite Difference Derivatives, and computations done with these definitions are said to be performed with the Finite Difference Method.

Please note that going forward, $\mathrm{d}x$ is a small but non-zero quantity, unless otherwise specified. We will also rely on the centered derivative definition because it provides a more accurate approximation (as will be demonstrated below), though these techniques apply equally to forward and backward derivative definitions with appropriate (and obvious) changes.

Taylor series

The Taylor series allows us to express the value of the function at a point x as an infinite sum of terms, where each term is expressed as a derivative at a point a where the function is infinitely differentiable.

$$f(x) = \sum_{n=0}^{\infty} \frac{\partial_x^n f(a)}{n!}(x-a)^n$$

The value of a function f at the point $x + \mathrm{d}x$ can thus be expressed as below, assuming that the function is infinitely differentiable at the point x.

$$f(x+\mathrm{d}x) = \sum_{n=0}^{\infty} \frac{1}{n!}\partial_x^n f\,\mathrm{d}x^n$$

This formulation helps estimate the numerical accuracy of our definitions, and also helps define Finite Difference Coefficients which can be used to express higher order derivatives and operators.

Estimating numerical accuracy

Using the Taylor series we can estimate the degree of accuracy of the numerical definition of the derivative. Let's start with the forward derivative.

$$\begin{aligned}
f(x+\mathrm{d}x) &= f(x) + \frac{f'(x)}{1!}\,\mathrm{d}x + \frac{f''(x)}{2!}\mathrm{d}x^2 + \cdots \\
f(x+\mathrm{d}x) - f(x) &= \frac{f'(x)}{1!}\,\mathrm{d}x + \frac{f''(x)}{2!}\mathrm{d}x^2 + \cdots \\
\frac{f(x+\mathrm{d}x) - f(x)}{\mathrm{d}x} &= f'(x) + \frac{f''(x)}{2!}\,\mathrm{d}x + \cdots \\
\frac{f(x+\mathrm{d}x) - f(x)}{\mathrm{d}x} &= f'(x) + \mathcal{O}(\mathrm{d}x)
\end{aligned}$$

The error term is of the order $\mathrm{d}x$. Now, let's estimate the numerical accuracy of the centered derivative.

$$\begin{aligned}
f(x+\mathrm{d}x) &= f(x) + \frac{f'(x)}{1!}\,\mathrm{d}x + \frac{f''(x)}{2!}\mathrm{d}x^2 + \cdots \\
f(x-\mathrm{d}x) &= f(x) - \frac{f'(x)}{1!}\,\mathrm{d}x + \frac{f''(x)}{2!}\mathrm{d}x^2 + \cdots \\
f(x+\mathrm{d}x) - f(x-\mathrm{d}x) &= 2\left[\frac{f'(x)}{1!}\,\mathrm{d}x + \frac{f'''(x)}{3!}\mathrm{d}x^3 \cdots\right] \\
\frac{f(x+\mathrm{d}x) - f(x-\mathrm{d}x)}{2\,\mathrm{d}x} &= f'(x) + \frac{f'''(x)}{3!}\mathrm{d}x^2 \cdots \\
\frac{f(x+\mathrm{d}x) - f(x-\mathrm{d}x)}{2\,\mathrm{d}x} &= f'(x) + \mathcal{O}(\mathrm{d}x^2)
\end{aligned}$$

The error term is now of the order $\mathrm{d}x^2$. This gives us an intuition for why the centered derivative definition is numerically more accurate than the forward/backward definitions.

Higher order derivatives

A second order derivative f'' can be defined in terms of the first order derivative f'.

$$f''(x) = \lim_{\mathrm{d}x \to 0} \frac{f'(x+\mathrm{d}x) - f'(x)}{\mathrm{d}x}$$

The numerical approximation for the 2^{nd} order derivative can then be expressed as

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

This expression can be derived from the Taylor series expansions for $f(x+\mathrm{d}x) + f(x-\mathrm{d}x)$ as laid out in the section above. Other higher order derivatives can also be similarly defined/derived.

Finite Difference Coefficients

Can we come up with a mathematical formula to compute the coefficients for these terms in order to define an arbitrary derivative?

Assume that a, b, c are unknown co-efficients (real numbers), and multiply the Taylor expansions of $f(x + \mathrm{d}x)$, $f(x)$ and $f(x - \mathrm{d}x)$ each with these co-efficients respectively.

$$\begin{aligned} af(x + \mathrm{d}x) &= a\left[f(x) + \frac{f'(x)}{1!}\,\mathrm{d}x + \frac{f''(x)}{2!}\mathrm{d}x^2 + \cdots\right] \\ bf(x) &= bf(x) \\ cf(x - \mathrm{d}x) &= c\left[f(x) - \frac{f'(x)}{1!}\,\mathrm{d}x + \frac{f''(x)}{2!}\mathrm{d}x^2 + \cdots\right] \end{aligned}$$

If we add these three equations together, we end up with

$$af(x + \mathrm{d}x) + bf(x) + cf(x - \mathrm{d}x) = f(x)\,[a + b + c] + \frac{f'(x)}{1!}\,\mathrm{d}x\,[a - c] + \cdots$$

Now, if we want the finite difference co-efficients for $f'(x)$, we can come up with the following system of linear equations which can be solved to yield the co-efficients.

$$\begin{aligned} a + b + c &= 0 \\ a - c &= \frac{1}{\mathrm{d}x} \\ a + c &= 0 \end{aligned}$$

This system of linear equations can then be expressed in matrix form as:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{\mathrm{d}x} \\ 0 \end{pmatrix}$$

Using matrix inverse, the co-efficients a, b, and c can be computed.

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 0 & 0.5 & 0.5 \\ 1 & 0 & -1 \\ 0 & -0.5 & 0.5 \end{pmatrix} \begin{pmatrix} 0 \\ \frac{1}{\mathrm{d}x} \\ 0 \end{pmatrix}$$

Performing this matrix multiplication yields the formula for the first derivative,

$$\frac{f(x + \mathrm{d}x) - f(x - \mathrm{d}x)}{2\,\mathrm{d}x} \approx f'(x)$$

The co-efficients for higher order derivatives can also be derived similarly.

Higher order operators

Until now, we've used what are called 3-point operators for defining the derivative. The 3 refers to the number of neighboring points around x that are considered in order to compute the derivative.

We can improve the numerical accurancy of our derivatives by taking into account a larger neighborhood around the point x. The intuition is that this should result in a higher order error term in the Taylor expansion, thus improving numerical accuracy.

For example, we can take into account a total of 5 neighboring points around x to compute the derivative: $f(x + 2\,\mathrm{d}x)$, $f(x + \mathrm{d}x)$, $f(x)$, $f(x - \mathrm{d}x)$ and $f(x - 2\,\mathrm{d}x)$. This is called a 5-point operator.

The finite difference co-efficients can be derived using the same technique described in the section above. We will just have a bigger matrix to invert, and more co-efficients to compute. For example, the 5-point operator definition for the first derivative of $f(x)$ will look like so:

$$f'(x) \approx \frac{\frac{1}{12}f(x - 2\,\mathrm{d}x) - \frac{2}{3}f(x - \mathrm{d}x) + \frac{2}{3}f(x + \mathrm{d}x) - \frac{1}{12}f(x + 2\,\mathrm{d}x)}{\mathrm{d}x}$$

This 5-point operator definition results in a numerical error term of degree $\mathcal{O}(\mathrm{d}x^4)$.

Computing finite difference co-efficients programmatically

point_range defines whether to calculate a 3-point derivative (-1:1), or a 5-point derivative (-2:2), or a 7-point derivative (-3:3) etc., and whether it should be a centered (-x:x)/forward(1:x)/backward (-x:-1) differential. We default to a 3-point operator.

It is immediately evident that we need a $(n + 1)$-point operator in order to compute the n^{th} derivative. In terms of the ranges defined above, $n + 1 = 2x$ for the centered derivative, $n = x + 1$ for the forward and backward derivatives.

point_range = -1:1

```
point_range = @bind point_range Slider(
    [
        # centered differential
        -1:1, -2:2, -3:3, -4:4,
        # forward differential
        1:1, 1:2, 1:3, 1:4,
        # backward differential
        -1:-1, -2:-1, -3:-1, -4:-1
    ], default=-1:1, show_value=true)
```

The co-efficients for the n^{th} derivative are given by the $(n+1)^{th}$ row of the matrix returned by the compute_coeffs() method defined below.

```
compute_coeffs (generic function with 1 method)
```

```
function compute_coeffs(point_range)
    ignore_order = length(point_range)
    coeffs = OffsetArray(Matrix{Float64}(undef, length(point_range), ignore_order), point_range, 0:(ignore_order-1))
    for row_idx ∈ point_range
        for col_idx ∈ 0:(ignore_order-1)
            coeffs[row_idx, col_idx] = (row_idx^col_idx) / factorial(col_idx)
        end
    end
    coeffs = convert(Matrix{Float64}, reshape(coeffs, (1:length(point_range)), 1:ignore_order))
    order_coeffs = inv(coeffs)
    order_coeffs
end
```

```
p = 3×1 Matrix{Float64}:
  1.0
 -2.0
  1.0
```

```
p = reshape(compute_coeffs(point_range)[3, :], length(point_range), 1) # This yields the co-efficients of the 2nd derivative for the chosen operator
```

Understanding the source function

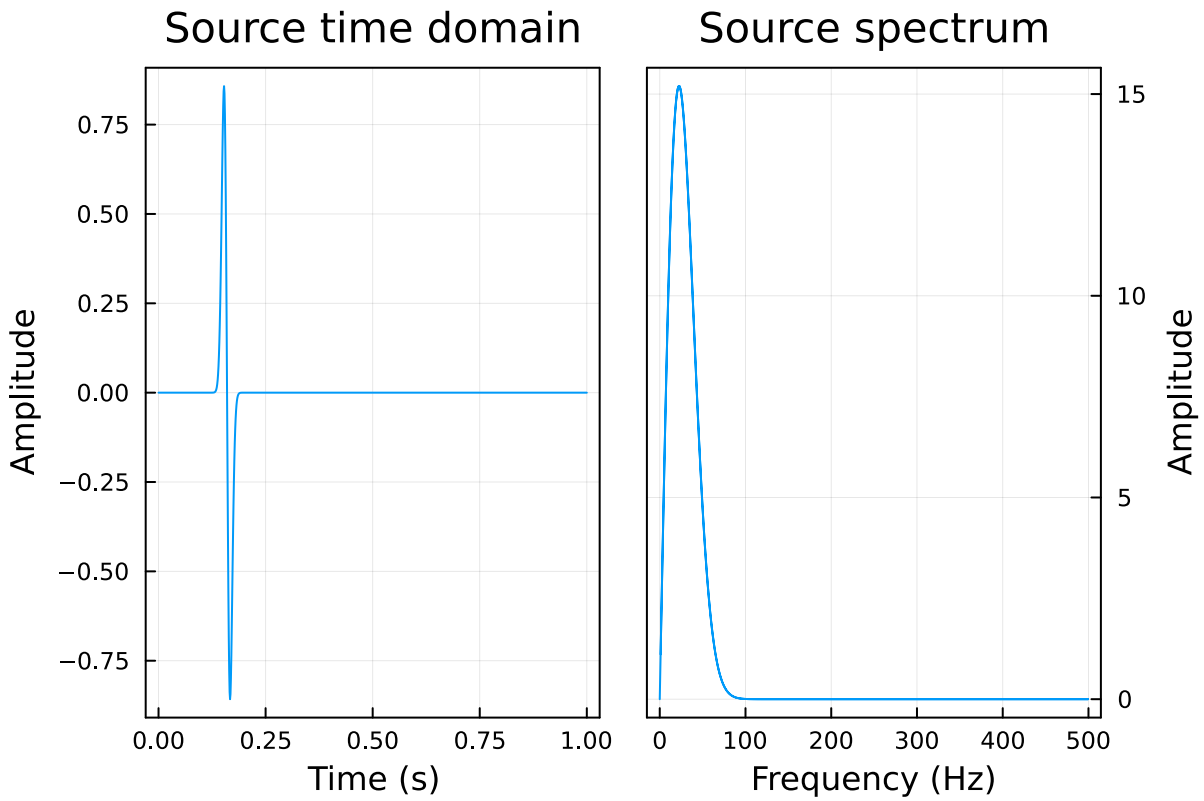
We will use the following source function $s(\mathbf{x}, t)$. It is the first derivative of a Gaussian pulse. We can define arbitrary source functions (periodic or not), and run the following simulations. Please note that changing this function will result in a re-run of all simulations below.

```
source (generic function with 1 method)
```

```
function source(f₀, t₀, times)
    -8 * f₀ * (times .- t₀) .* exp.(-(times .- t₀).^2 * (4 * f₀).^2)
end
```

```
source_visualization (generic function with 1 method)
```

```
function source_visualization()
    # Physical domain
    xmax = 10000
    c₀ = 343 # Velocity of wave in the medium. 343 m/s

    # Source
    f₀ = 25 # Dominant frequency of the source
    src_pos = 0.50 # The physical location of the source as a percentage of xmax

    # Simulation parameters
    tmax = 1000 # Maximum number of time steps
    nλ = 20 # Increase this value to reduce numerical anisotropy

    # Derived parameters
    λ = c₀/f₀ # Dominant wavelength
    dx = λ/nλ # Grid spacing along the x-axis
    dt = 0.001 # Uncomment this line to increase numerical anisotropy
    t₀ = 0.16 # Time in seconds when the source pulse is generated
    times = 0:dt:(dt*tmax)
    xpoints = 1:dx:xmax
    num_grid_points = length(xpoints)

    # The source function which generates the disturbance
    src = -8 * f₀ * (times .- t₀) .* exp.(-(times .- t₀).^2 * (4 * f₀).^2)

    p1 = plot(times, src, label=nothing, title="Source time domain", xlabel="Time
    (s)", ylabel="Amplitude", framestyle=:box)
    signal = src
    amp = fft(signal)
    freq = fftfreq(length(times), 1/dt)
    p2 = plot(abs.(freq), abs.(amp), label=nothing, title="Source spectrum",
    xlabel="Frequency (Hz)", ylabel="Amplitude", ymirror=true, framestyle=:box)
    plot(p1, p2, layout=2)
end
```




```
1 source_visualization()
```

Solving the wave equation

Now, we are ready to solve the wave equation numerically. Let's start with the Acoustic wave equation.

$$\partial_t^2 u(\mathbf{x}, t) = c(\mathbf{x})^2 \cdot \partial_x^2 u(\mathbf{x}, t) + s(\mathbf{x}, t)$$

1D wave equation

The 2nd order time derivative at time t can be numerically expressed using the centered finite difference method as:

$$\partial_t^2 u(x,t) \approx \frac{u(x,t+dt) - 2u(x,t) + u(x,t-dt)}{dt^2}$$

Similarly, the numerical 2nd order position derivative at position x can be expressed as:

$$\partial_x^2 u(x,t) \approx \frac{u(x+dx,t) - 2u(x,t) + u(x-dx,t)}{dx^2}$$

An alternative and more concise notation is $u(x,t) = u_x^t$. In this notation, the subscript represents the position and the superscript represents time.

In this notation, the numerical 2nd order derivative for time can be expressed as:

$$\partial_t^2 u \approx \frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{dt^2}$$

where, $t = n\,\mathrm{d}t$ and $x = i\,\mathrm{d}x$.

A similar expression exists for the numerical 2nd order position derivative.

The 1D wave equation can then be expressed as:

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{dt^2} \approx c_i^2 \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{dx^2} + s_i^n$$

We can now extrapolate the value of u_i^{n+1} in terms of all other variables, and this will yield the displacement caused by the wave in the future as a function of current and past displacements. This in turn allows us to numerically compute and visualize the wave as it propagates through the medium.

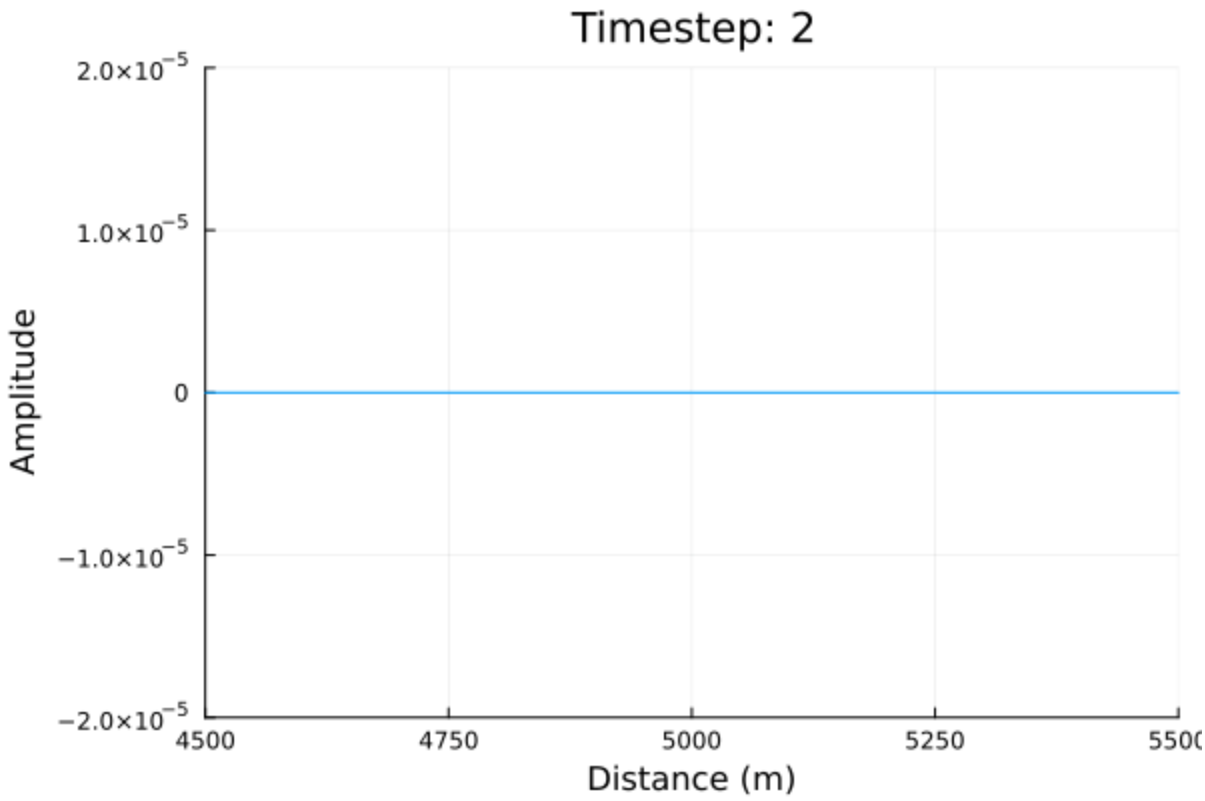
Modeling the physical domain

We will now use the Finite Difference Method to simulate wave propagation in a homogeneous medium in one dimension.

Let us assume that we have a homogeneous medium of length L = $10,000m$, in which the wave propagates with a speed $c(x) = 343\mathrm{m/s}$. We start our simulation at time 0, and there is a disturbance source at time t_0 with a dominant frequency of f_0 (as demonstrated above)

```
with_numerical_anisotropy_1D (generic function with 1 method)
```

```
function with_numerical_anisotropy_1D()
    # Physical domain
    xmax = 10000
    c₀ = 343 # Velocity of wave in the medium. 343 m/s

    # Source
    f₀ = 25 # Dominant frequency of the source
    src_pos = 0.50 # The physical location of the source as a percentage of xmax

    # Simulation parameters
    tmax = 1000 # Maximum number of time steps
    nλ = 20 # Increase this value to reduce numerical anisotropy

    # Derived parameters
    λ = c₀/f₀ # Dominant wavelength
    dx = λ/nλ # Grid spacing along the x-axis
    dt = 0.001 # Uncomment this line to increase numerical anisotropy
    t₀ = 0.16 # Time in seconds when the source pulse is generated
    times = 0:dt:(dt*tmax)
    xpoints = 1:dx:xmax
    num_grid_points = length(xpoints)

    # The source function which generates the disturbance
    src = source(f₀, t₀, times)

    # Numerical computation
    fⁿⱼ = zeros(num_grid_points)
    fⁿ⁻¹ⱼ = zeros(num_grid_points)
    fⁿ⁺¹ⱼ = zeros(num_grid_points)
    ∂ₓ²f = zeros(num_grid_points)
    s = Int(ceil(src_pos*num_grid_points)) # Index of source signal

    # Obtain the co-efficients for the 2nd derivative for the arbitrary n-point
    operator
    coeffs = reshape(compute_coeffs(point_range)[3, :], length(point_range), 1)
    matrix = Matrix(hcat([copy(fⁿⱼ) for i in 1:length(point_range)]...)) * coeffs

    anim = @animate for n in 2:(length(times)-1)
        for j in 2:(num_grid_points-1)
            ∂ₓ²f[j] = (fⁿⱼ[j+1] - (2*fⁿⱼ[j]) + fⁿⱼ[j-1])/(dx^2) * c₀^2 * dt^2
            #∂ₓ²f[j] = (matrix[j, 1])/(dx^2) * c₀^2 * dt^2
        end
        fⁿ⁺¹ⱼ = (∂ₓ²f) .+ (2 .* fⁿⱼ) .- fⁿ⁻¹ⱼ
        fⁿ⁺¹ⱼ[s] = fⁿ⁺¹ⱼ[s] + src[n] / (dx) * dt^2
        fⁿⱼ, fⁿ⁻¹ⱼ = fⁿ⁺¹ⱼ, fⁿⱼ
        plot(xpoints, fⁿⱼ, title="Timestep: $(n)", label=nothing, xlims=(4500,
        5500), ylabel="Amplitude", xlabel="Distance (m)", ylims=(-2e-5, 2e-5))
    end

    gif(anim, fps=60)
end
```




```
Saved animation to /var/folders/7k/r1g1zhmx16gbtm1hbllhhk_w0000gs/T/jl_g32tKTeTS5.gif
```

Numerical anisotropy

What are the squiggly lines that we see in the visualization above? They are the consequence of errors due to our numerical approximations, and this phenomena is called Numerical Anisotropy.

We obviously want to avoid numerical anisotropy, but in order to do that, we need to understand the source of these errors. And we use an analysis technique developed by Von Neumann to do that.

For

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{dt^2} \approx c_i^2 \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{dx^2} + s_i^n$$

For plane waves, u_i^n takes the form $e^{j(\mathbf{kx}-\omega t)}$, where $j = \sqrt{-1}$. Substituting that solution form into the discrete version of the wave propagation results in the following (assume 1D for ease of analysis):

$$e^{j(ki\,\mathrm{d}x\, -\omega n\,\mathrm{d}t)} \left[\frac{e^{j\omega\,\mathrm{d}t} - 2 + e^{-j\omega\,\mathrm{d}t}}{\mathrm{d}t^2}\right] \approx (c_i^n)^2 e^{j(ki\,\mathrm{d}x\, -\omega n\,\mathrm{d}t)} \left[\frac{e^{jk\,\mathrm{d}x} - 2 + e^{-jk\,\mathrm{d}x}}{\mathrm{d}x^2}\right] + s_i^n$$

If we neglect the source term s_i^n, and manipulate this equation further using Euler's relation $e^{jx} = \cos(x) + j\sin(x)$, and some trigonometric identities, we end up with the following dispersion relation.

$$\sin(\omega\frac{\mathrm{d}t}{2}) = c\frac{dt}{dx}\sin(k\frac{\mathrm{d}x}{2})$$

Please note that if we were using the analytical form or the continuous version, a similar analysis yields the well known dispersion relation $c = \frac{\omega}{k}$.

CFL criterion

In order for the numerical dispersion relation $\sin(\omega\frac{\mathrm{d}t}{2}) = c\frac{dt}{dx}\sin(k\frac{\mathrm{d}x}{2})$ to have a solution, the criteria that needs to be satisfied is that $c\frac{\mathrm{d}t}{\mathrm{d}x} \leq 1$.

This condition is called the Courant-Friedrichs-Lewy criterion (a.k.a, the CFL criterion) for numerical analysis. It determines the relationship between the time increment $\mathrm{d}t$ and the grid increment $\mathrm{d}x$.

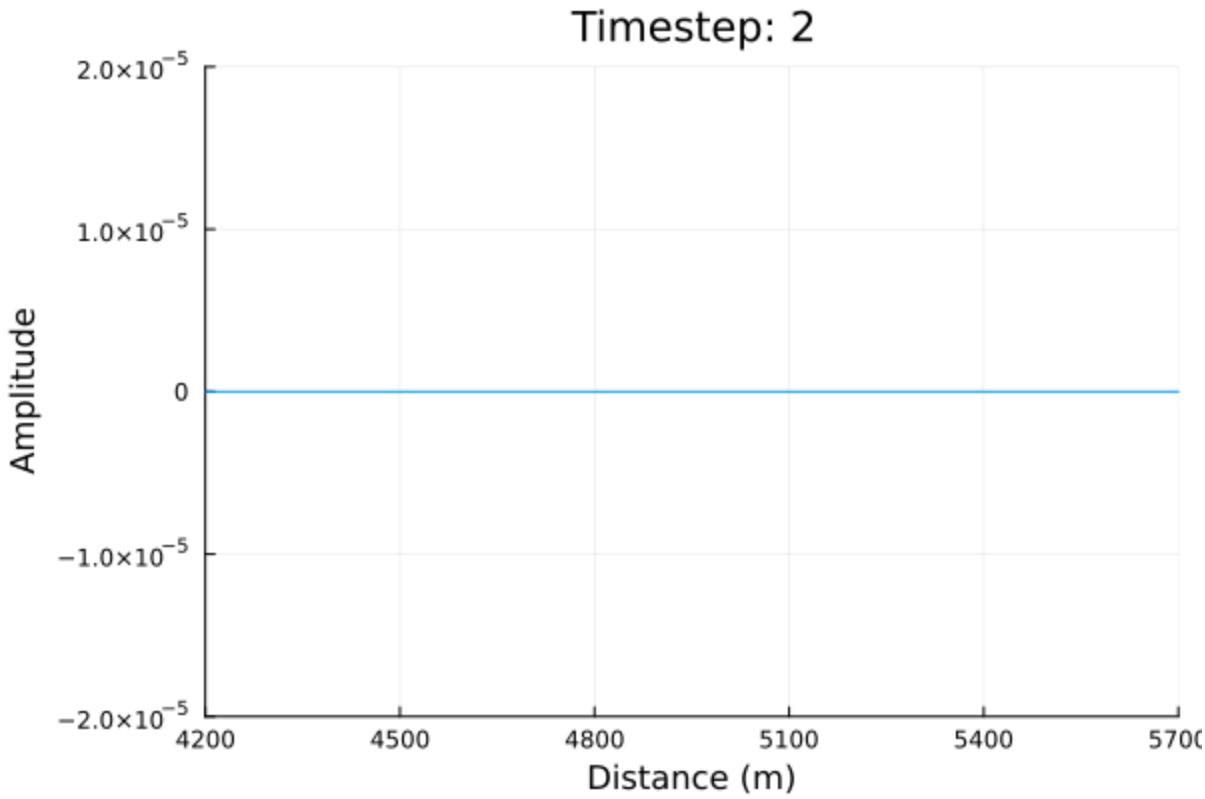
It should be noted that this is not a sufficient condition for a numerical solution in itself.

Using the CFL criteria

```
with_cfl_criteria_1D (generic function with 1 method)
```

```
function with_cfl_criteria_1D()
    # Physical domain
    xmax = 10000
    c₀ = 343 # Velocity of wave in the medium. 343 m/s
    ϵ = 1 # From CFL criteria. Setting this value to be <1, results in some numerical anisotropy. Setting it to >1 results in solution explosion. Setting to 1 should result in a solution that is very similar to the analytical/true solution.

    # Source
    f₀ = 25 # Dominant frequency of the source
    src_pos = 0.50 # The physical location of the source as a percentage of xmax

    # Simulation parameters
    tmax = 1000 # Maximum number of time steps
    nλ = 20 # Increase this value to reduce numerical anisotropy

    # Derived parameters
    λ = c₀/f₀ # Dominant wavelength
    dx = λ/nλ # Grid spacing along the x-axis
    dt = dx/c₀ * ϵ
    t₀ = 0.16 # Time in seconds when the source pulse is generated
    times = 0:dt:(dt*tmax)
    xpoints = 1:dx:xmax
    num_grid_points = length(xpoints)

    # The source function which generates the disturbance
    src = source(f₀, t₀, times)

    # Numerical computation
    fⁿⱼ = zeros(num_grid_points)
    fⁿ⁻¹ⱼ = zeros(num_grid_points)
    fⁿ⁺¹ⱼ = zeros(num_grid_points)
    ∂ₓ²f = zeros(num_grid_points)
    s = Int(ceil(src_pos*num_grid_points)) # Index of source signal

    # Obtain the co-efficients for the 2nd derivative for the arbitrary n-point operator
    coeffs = reshape(compute_coeffs(point_range)[3, :], length(point_range), 1)
    matrix = Matrix(hcat([copy(fⁿⱼ) for i in 1:length(point_range)]...)) * coeffs

    anim = @animate for n in 2:(length(times)-1)
        for j in 2:(num_grid_points-1)
            ∂ₓ²f[j] = (fⁿⱼ[j+1] - (2*fⁿⱼ[j]) + fⁿⱼ[j-1])/(dx^2) * c₀^2 * dt^2
            #∂ₓ²f[j] = (matrix[j, 1])/(dx^2) * c₀^2 * dt^2
        end
        fⁿ⁺¹ⱼ = (∂ₓ²f) .+ (2 .* fⁿⱼ) .- fⁿ⁻¹ⱼ
        fⁿ⁺¹ⱼ[s] = fⁿ⁺¹ⱼ[s] + src[n] / (dx) * dt^2
        fⁿⱼ, fⁿ⁻¹ⱼ = fⁿ⁺¹ⱼ, fⁿⱼ
        plot(xpoints, fⁿⱼ, title="Timestep: $(n)", label=nothing, xlims=(4200, 5700), ylabel="Amplitude", xlabel="Distance (m)", ylims=(-2e-5, 2e-5))
    end

    gif(anim, fps=60)
```

```
    end
```




```
with_cfl_criteria_1D()
```

```
Saved animation to /var/folders/7k/r1g1zhmx16gbtm1hbllhhk_w0000gs/T/jl_qFkF8FA1yZ.gif
```

Wait, why is the x-axis different from the visualization with numerical anisotropy? Because the $\mathrm{d}t$ is different (slightly larger) due to the CFL criteria, which means that the wave propagates out a little further than the visualization with numerical anisotropy.

2D wave equation

The 2D wave equation has a similar formulation as the 1D wave equation with more indices. $u(x, y, t)$ is denoted as $u^n_{i,j}$, where $x = i\,\mathrm{d}x$, $y = j\,\mathrm{d}y$, and $t = n\,\mathrm{d}t$.

$$\frac{u^{n+1}_{i,j} - 2u^n_{i,j} + u^{n-1}_{i,j}}{dt^2} \approx c^2_{i,j}\left[\frac{u^n_{i+1,j} - 2u^n_{i,j} + u^n_{i-1,j}}{dx^2} + \frac{u^n_{i,j+1} - 2u^n_{i,j} + u^n_{i,j-1}}{dy^2}\right] + s^n_{i,j}$$

We use the same extrapolation scheme as the 1D case to compute the displacement caused by the wave as a function of the current and previous timestep displacements.

Modeling the physical domain

We will now use the Finite Difference Method to simulate wave propagation in a homogeneous medium in two dimensions.

Let us assume that we have a homogeneous medium of length L = $500 \times 500m^2$, in which the wave propagates with a speed $c(x) = 580\mathrm{m/s}$. We start our simulation at time 0, and there is a disturbance source at time t_0 with a dominant frequency of f_0.

w:

U

t

C

c

w:

I

[1
h
/
/
wa
s:

[2
h
/\

[3
h
/\