Numerical Simulation of the 1D Wave Equation using the Finite Difference Method

Mina Shiri (40011023)

July 16, 2025

Abstract

This report details the numerical solution of the one-dimensional wave equation using the Finite Difference Method (FDM). The simulation is implemented in JavaScript and visualized in real-time on an HTML canvas. The problem considers a 1D domain with homogeneous Dirichlet boundary conditions, a variable wave propagation speed, and an external excitation source in the form of a temporary pulse. We discuss the mathematical formulation, the discretization using central differences, the stability conditions for the numerical scheme, and the implementation details of the interactive simulation. The results show the successful propagation of the wave, including reflection and transmission phenomena at the interface of differing wave speeds.

1 Introduction

The wave equation is a fundamental second-order linear partial differential equation (PDE) that describes the propagation of a variety of waves, such as sound waves, light waves, and vibrations in a string. For a one-dimensional domain, the equation is given by:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$

where $u(x,t)$ is the amplitude of the wave at position x and time t, and c is the constant speed of wave propagation. In this project, we explore a more complex scenario where the wave speed $c(x)$ is a function of position.

2 Problem Statement

We aim to solve the 1D wave equation on a spatial domain $x \in [0,1]$ for a time interval $t \in [0,3]$. The specific conditions for the problem are as follows:

- **Domain:** Length $L = 1$ meter.
- **Boundary Conditions:** Homogeneous Dirichlet boundary conditions are applied at both ends of the domain, meaning the wave amplitude is fixed at zero.

$$u(0,t) = 0 \quad \text{and} \quad u(1,t) = 0$$

- **Initial Conditions:** The system starts from rest.

$$u(x,0) = 0 \quad \text{and} \quad \frac{\partial u}{\partial t}(x,0) = 0$$

- **Wave Propagation Speed:** The speed of the wave is not constant across the domain. It is defined as:

$$c(x) = \begin{cases} 1\,\text{m/s} & \text{for } 0 \le x \le 0.75 \\ 4\,\text{m/s} & \text{for } 0.75 < x \le 1.0 \end{cases}$$

- **Excitation Source:** The system is excited by an external pulse function $F(t)$ at a specific point $x_e = 0.5$. The pulse is active for a finite duration.

$$u(0.5,t) = F(t) = \begin{cases} \sin\left(\frac{\pi(t-0.1)}{0.5}\right) & \text{for } 0.1 \le t < 0.6 \\ 0 & \text{otherwise} \end{cases}$$

3 Numerical Method

To solve the wave equation numerically, we employ the Finite Difference Method (FDM), which involves discretizing the continuous domain into a grid of points and approximating the derivatives with finite differences.

3.1 Discretization

We discretize the spatial domain into N_x points with a step size of $\Delta x = L/(N_x - 1)$, and the time domain with a step size of Δt. The wave function $u(x, t)$ is then represented by $u_i^n = u(i\Delta x, n\Delta t)$.

3.2 Central Difference Approximation

We use second-order central difference approximations for both the time and space derivatives:

$$\frac{\partial^2 u}{\partial t^2} \approx \frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{(\Delta t)^2}$$

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

Substituting these approximations into the wave equation gives the explicit finite difference scheme:

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{(\Delta t)^2} = c_i^2 \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2}$$

Rearranging to solve for the wave function at the next time step, u_i^{n+1}, we get:

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

This equation allows us to compute the state of the wave at time $n + 1$ using the states at times n and $n - 1$.

3.3 Stability Condition (CFL)

For the explicit scheme to be numerically stable, the time step Δt must satisfy the Courant-Friedrichs-Lewy (CFL) condition:

$$\frac{c_{\max} \Delta t}{\Delta x} \leq 1$$

where $c_{\max}$ is the maximum wave speed in the domain. In our case, $c_{\max} = 4$ m/s. The implementation sets the time step to be 95% of the stability limit to ensure robustness.

4 Implementation Details

The simulation is built using HTML, CSS, and JavaScript, making it interactive.

- **HTML/CSS:** A user interface is created with a canvas element for visualization, control buttons (Start/Pause, Reset), and informational displays for the simulation time.
- **JavaScript:** The core logic resides in a JavaScript script.
  - **Parameters:** Key simulation parameters such as L, T_{MAX}, NX, DX, DT, and the excitation details are defined as constants.
  - **Data Structures:** Three arrays (`u_prev`, `u_curr`, `u_next`) are used to store the wave amplitude at the previous, current, and next time steps.
  - **Initialization:** An `initialize()` function sets up the grid, defines the variable wave speed array c, pre-calculates the coefficient $(c_i \Delta t/\Delta x)^2$ for efficiency, and resets all state variables.

- **Main Loop:** A `simulationLoop()` function is repeatedly called using `requestAnimationFrame`. In each iteration, it calculates `u_next` for all interior points based on the finite difference equation, applies the excitation pulse at the specified location and time, and then updates the `u_prev` and `u_curr` arrays for the next iteration.
- **Boundary Conditions:** The Dirichlet boundary conditions are enforced by ensuring that the first and last elements of the wave function arrays remain zero.
- **Visualization:** A `draw()` function clears and redraws the wave on the canvas at each time step, providing a real-time animation of the wave's motion.

5 Simulation and Results

Running the simulation reveals several key wave phenomena:

1. **Excitation:** At $t = 0.1$s, a pulse begins to form at $x = 0.5$. This pulse grows and splits into two waves traveling in opposite directions.
2. **Propagation:** The two waves propagate towards the boundaries. The wave traveling left moves at $c = 1$ m/s, while the wave traveling right also moves at $c = 1$ m/s until it reaches the interface.
3. **Reflection at Boundaries:** When the waves reach the fixed boundaries at $x = 0$ and $x = 1$, they reflect and invert their phase, as expected from Dirichlet boundary conditions.
4. **Transmission and Reflection at Interface:** When the right-traveling wave reaches the interface at $x = 0.75$, it is partially reflected and partially transmitted. The transmitted wave enters the region with $c = 4$ m/s and visibly speeds up, demonstrated by an increase in its wavelength. The reflected portion inverts its direction and travels back towards the center at the original speed of $c = 1$ m/s.

The visual animation provides an intuitive understanding of how waves behave in a medium with non-uniform properties.

6 Conclusion

This project successfully demonstrates the numerical solution of the 1D wave equation with complex conditions using the finite difference method. The implementation in JavaScript provides an effective and interactive tool for visualizing wave dynamics. The simulation accurately captures fundamental wave behaviors, including propagation, reflection, and transmission, offering valuable insight into the underlying physics.

A JavaScript Simulation Code

```
function runWaveSimulation() {
    // --- Get DOM Elements ---
    const canvas = document.getElementById('waveCanvas');
    const ctx = canvas.getContext('2d');
    const timeValueEl = document.getElementById('timeValue');
    const startStopBtn = document.getElementById('startStopBtn');
    const resetBtn = document.getElementById('resetBtn');

    // --- Simulation Parameters ---
    const L = 1.0;            // Length of the domain [m]
    const T_MAX = 3.0;        // Total simulation time [s]
    const NX = 201;           // Number of spatial grid points
    const DX = L / (NX - 1); // Spatial step size
    const MAX_SPEED = 4.0;  // Maximum wave speed in the domain [m/s]

    // --- Stability (CFL Condition) ---
    const DT = (DX / MAX_SPEED) * 0.95; // Time step size [s]
    const NT = Math.floor(T_MAX / DT);  // Total number of time steps

    // --- Excitation Pulse Parameters ---
    const PULSE_START_TIME = 0.1;
    const PULSE_DURATION = 0.5;
    const PULSE_END_TIME = PULSE_START_TIME + PULSE_DURATION;
    const EXCITATION_POS_X = 0.5;
    const EXCITATION_INDEX = Math.round(EXCITATION_POS_X / DX);

    // --- Wave Function Arrays ---
    let u_prev, u_curr, u_next;

    // --- Wave Speed Array ---
    const c = new Float32Array(NX);
    const c_squared_dt_squared_over_dx_squared = new Float32Array(NX);

    // --- Animation State Variables ---
    let animationFrameId;
    let isRunning = false;
    let timeStep = 0;

    function initialize() {
        // ... (Initialization logic)
    }

    function simulationLoop() {
        if (!isRunning || timeStep >= NT) {
            // ... (Stop condition)
            return;
        }

        // --- Core Finite Difference Calculation ---
        for (let i = 1; i < NX - 1; i++) {
            const laplacian_u = u_curr[i + 1] - 2 * u_curr[i] + u_curr[i - 1];
            u_next[i] = 2 * u_curr[i] - u_prev[i] + c_squared_dt_squared_over_dx_squared
    [i] * laplacian_u;
        }

        // --- Apply Excitation Pulse ---
        const currentTime = timeStep * DT;
        if (currentTime >= PULSE_START_TIME && currentTime < PULSE_END_TIME) {
            const pulse_time = currentTime - PULSE_START_TIME;
            u_next[EXCITATION_INDEX] = Math.sin( (pulse_time / PULSE_DURATION) * Math.PI
     );
        }

        // --- Update arrays for the next time step ---
        u_prev = u_curr;
        u_curr = u_next;
        u_next = new Float32Array(NX).fill(0);

        // --- Draw and advance time ---
        draw();
        timeValueEl.textContent = currentTime.toFixed(3);
        timeStep++;
        animationFrameId = requestAnimationFrame(simulationLoop);
    }

    function draw() {
        // ... (Canvas drawing logic)
    }

    // --- Event Listeners & Initial Call ---
    startStopBtn.addEventListener('click', () => { /* ... */ });
    resetBtn.addEventListener('click', initialize);
    initialize();
}

runWaveSimulation();
```

Listing 1: Core JavaScript code for the wave equation simulation.