Computational Modeling
Project 5

Maroua Ouahbi
Constructor University

INTRODUCTION

The wave equation is a fundamental equation in physics used to describe the propagation of waves through a medium. For a vibrating string, the wave equation is given by:

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

where *u(x,t)* represents the displacement of the string at position x and time t, and cc is the wave speed along the string. This equation assumes that the string is perfectly flexible, has uniform tension, and that no external forces are acting on it. The wave equation describes the motion of the string as a function of both time and space.

To solve this equation numerically, a discretization of both the space and time domains is required. In this project, the leapfrog algorithm is used to solve the wave equation. The leapfrog method is a simple but effective approach where the displacement at the next time step is calculated based on the current and previous displacements. This algorithm alternates between computing displacement and velocity.

The leapfrog update equation is:

$$u_i^{j+1} = 2u_i^j - u_i^{j-1} + c'^2(u_{i+1}^j + u_{i-1}^j - 2u_i^j)$$

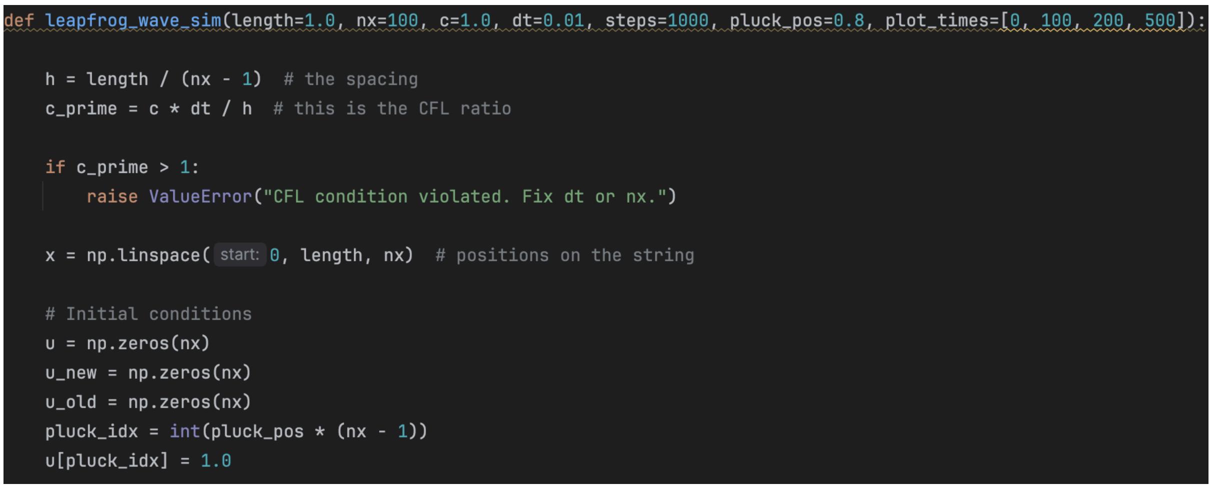
Where u_i^j is the displacement of the string at position i and time step j, \(c' = \frac{c \Delta t}{h} \) is the numerical velocity, with h being the wave speed, Δt the time step, and h the grid spacing. The equation predicts the displacement at the next time step using the current and neighboring displacements.

This method is unconditionally stable when the numerical parameters are chosen appropriately, making it well-suited for simulating wave motion on a string, particularly under periodic boundary conditions where the string ends are fixed.

CODE EXPLANATION

1. Initialization

The first part of the code defines the grid size and the time step. The string is initially "plucked" near one end, represented by a bell-shaped curve. The displacement is set to this initial shape, and the velocity is set to zero.



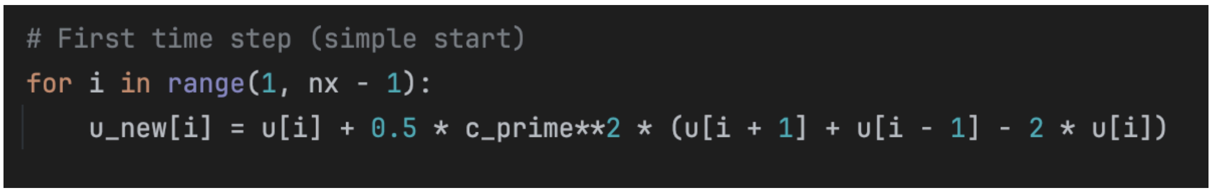
```python
def leapfrog_wave_sim(length=1.0, nx=100, c=1.0, dt=0.01, steps=1000, pluck_pos=0.8, plot_times=[0, 100, 200, 500]):

    h = length / (nx - 1)  # the spacing
    c_prime = c * dt / h  # this is the CFL ratio

    if c_prime > 1:
        raise ValueError("CFL condition violated. Fix dt or nx.")

    x = np.linspace( start: 0, length, nx)  # positions on the string

    # Initial conditions
    u = np.zeros(nx)
    u_new = np.zeros(nx)
    u_old = np.zeros(nx)
    pluck_idx = int(pluck_pos * (nx - 1))
    u[pluck_idx] = 1.0
```

The grid is created with n_x grid points. The grid spacing h is calculated from the number of grid points.

The string is given an initial displacement near one end, modeled by a Gaussian function. The initial velocity is set to zero.

2. First Time Step

At the first time step, the displacement for the next time step is calculated based on the leapfrog method. This update uses the current displacement and its neighbors.

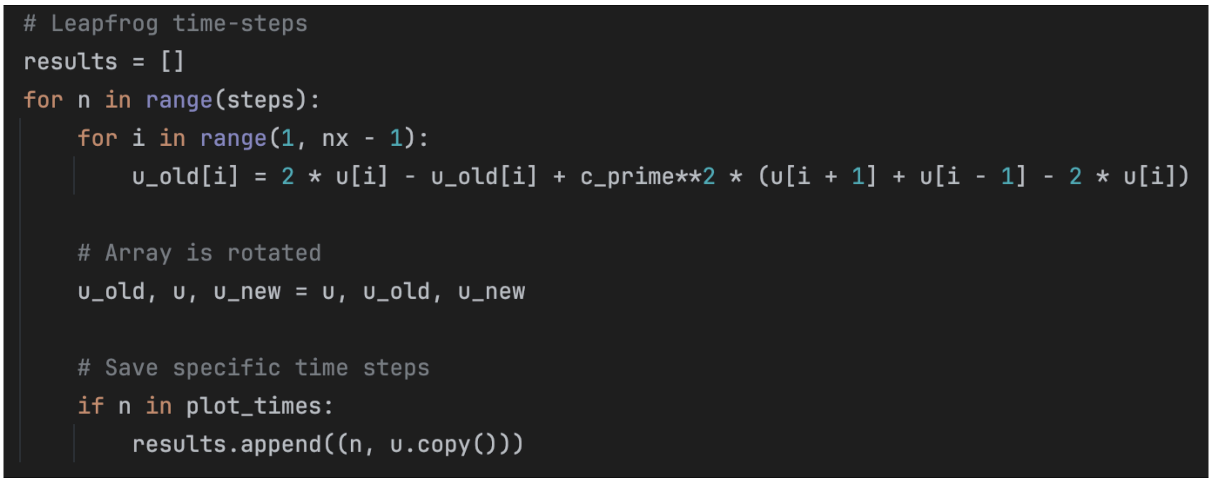


```python
# First time step (simple start)
for i in range(1, nx - 1):
    u_new[i] = u[i] + 0.5 * c_prime**2 * (u[i + 1] + u[i - 1] - 2 * u[i])
```

The leapfrog method is applied to calculate the displacement for the next time step using the current values.

3. Time-Stepping Loop

This part of the code runs the main simulation. For each time step, the displacement is updated, and the previous, current, and future displacements are rotated to prepare for the next time step.

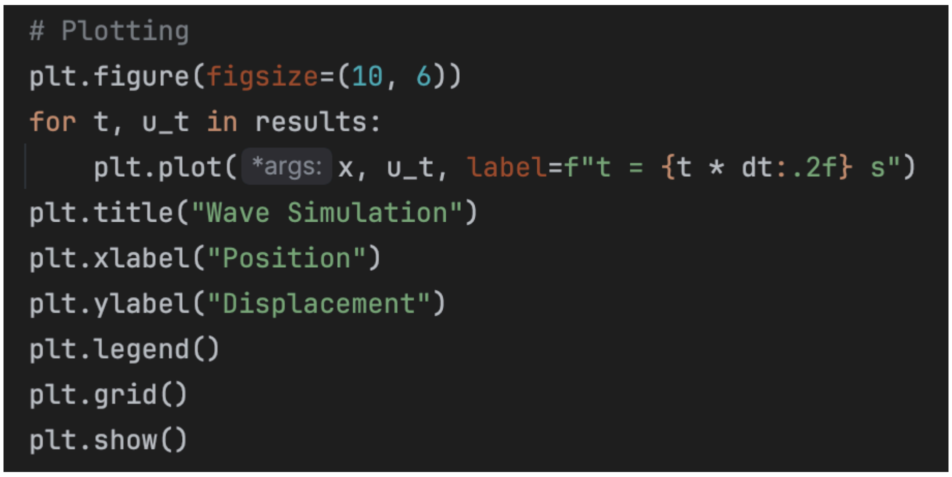


```python
# Leapfrog time-steps
results = []
for n in range(steps):
    for i in range(1, nx - 1):
        u_old[i] = 2 * u[i] - u_old[i] + c_prime**2 * (u[i + 1] + u[i - 1] - 2 * u[i])

    # Array is rotated
    u_old, u, u_new = u, u_old, u_new

    # Save specific time steps
    if n in plot_times:
        results.append((n, u.copy()))
```

- **Time-Stepping**In each loop iteration, the displacement for the next time step is calculated for the interior points of the string, using the leapfrog formula.
- **Rotation**: The past, present, and future displacements are rotated after each time step to keep the method going.
- **Results Saving**: The displacement at specific time steps is saved for later plotting.

4. Plotting Results

After the simulation is completed, the code plots the displacement of the string at the saved time steps, showing how the wave evolves.



```python
# Plotting
plt.figure(figsize=(10, 6))
for t, u_t in results:
    plt.plot( *args: x, u_t, label=f"t = {t * dt:.2f} s")
plt.title("Wave Simulation")
plt.xlabel("Position")
plt.ylabel("Displacement")
plt.legend()
plt.grid()
plt.show()
```

The results saved for each specified time step are plotted. The displacement of the string is shown at different times, illustrating the wave's motion.

5. Running Simulations with Different Grids

Finally, the simulation is run twice: once with a regular grid and again with a finer grid. This comparison highlights the effect of grid resolution on the accuracy and smoothness of the wave.

```
# Simulation with two different grids to be compared
leapfrog_wave_sim(nx=100, dt=0.01, steps=500, plot_times=[0, 50, 100, 200])
leapfrog_wave_sim(nx=200, dt=0.005, steps=500, plot_times=[0, 50, 100, 200])
```

The first simulation uses a coarser grid with fewer points, while the second uses a finer grid with more points and a smaller time step. The results from both simulations are compared to show how grid resolution affects the wave's accuracy.

COMPARISON & OBSERVATIONS

The key observations from the simulations are as follows:

1. **Wave Reflection**: The wave propagates across the string and reflects off the fixed ends, continuing its motion in the opposite direction after bouncing back.
2. **Effect of Grid Resolution**:
   - With the regular grid, the wave exhibits a more jagged appearance, with fewer grid points providing a rougher representation of the wave's motion.
   - The finer grid results in a smoother wave, as more grid points are used to better approximate the continuous motion of the wave.
3. **Time Step Size**: A smaller time step allows the wave to be more accurately simulated, particularly with a finer grid, improving the precision of the wave's propagation.

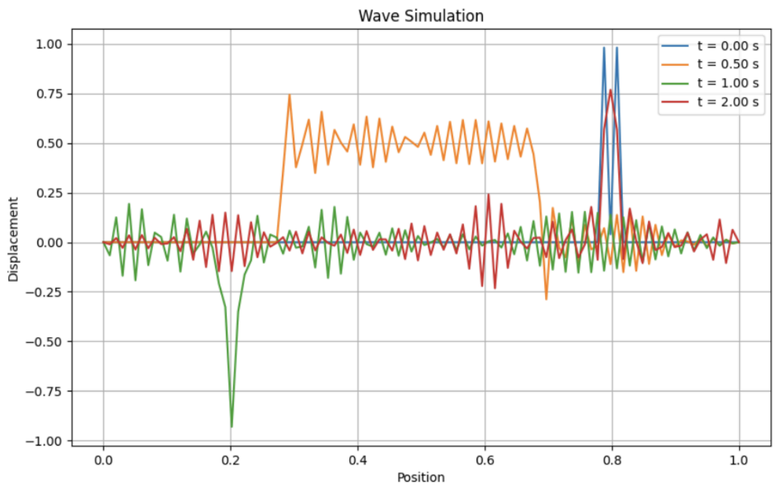


Figure 1: *Wave simulation with regular grid.*

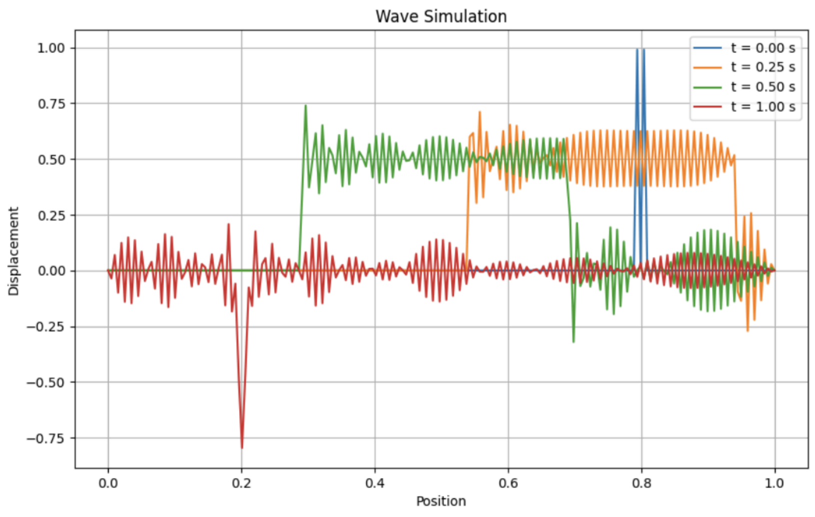


Figure 2: *Wave simulation with finer grid.*

The first graph shows the wave propagation at different time steps using the normal grid, while the second graph displays the results using the finer grid. The differences in the smoothness and accuracy of the wave are clearly visible when comparing the two grids.

CONCLUSION

The leapfrog algorithm successfully models the wave equation for the motion of a vibrating string. The simulations provided valuable insights into how the wave propagates and reflects at the boundaries. The comparison between simulations with different grid resolutions demonstrated the importance of grid spacing for accuracy. A finer grid provides a more accurate and smoother representation of the wave, highlighting the significance of resolution in numerical simulations. The leapfrog method proved to be an efficient and reliable technique for solving the wave equation, offering a clear understanding of wave dynamics on a string.