Lab 5

Animating the Wave Equation: Staggered Leapfrog

In the last couple of labs we handled the wave equation by Fourier analysis, turning the partial differential equation into a set of ordinary differential equations using separation of variables.[1] But separating the variables and expanding in orthogonal functions is not the only way to solve partial differential equations, and in fact in many situations this technique is awkward, ineffective, or both. In this lab we will study another way of solving partial differential equations using a spatial grid and stepping forward in time. As an added attraction, this method automatically supplies a beautiful animation of the solution. There are several algorithms of this type that can be used on wave equations, so this is just an introduction to a larger subject. The method we will show you here is called *staggered leapfrog*; it is the simplest good method that we know.

The wave equation with staggered leapfrog

Consider again the classical wave equation with wave speed c.

$$\frac{\partial^2 y}{\partial t^2} - c^2 \frac{\partial^2 y}{\partial x^2} = 0 \tag{5.1}$$

For a string, the wave speed is related to tension and density via $c = \sqrt{T/\mu}$. The boundary conditions are usually either of *Dirichlet* type (values specified):

$$y(0,t) = f_{\text{left}}(t) \quad ; \quad y(L,t) = f_{\text{right}}(t) \tag{5.2}$$

or of *Neumann* type (derivatives specified):

$$\frac{\partial y}{\partial x}(0) = g_{\text{left}}(t) \quad ; \quad \frac{\partial y}{\partial x}(L) = g_{\text{right}}(t) \tag{5.3}$$

Sometimes mixed boundary conditions specify a relation between the value and derivative, as at the bottom of the hanging chain. In any case, some set of conditions at the endpoints are required to solve the wave equation. It is also necessary to specify the initial state of the string, giving its starting position and velocity as a function of position:

$$y(x, t=0) = y_0(x) \quad ; \quad \frac{\partial y(x,t)}{\partial t}|_{t=0} = v_0(x) \tag{5.4}$$

[1] N. Asmar, *Partial Differential Equations and Boundary Value Problems* (Prentice Hall, New Jersey, 2000), p. 87-110.

Both of these initial conditions are necessary because the wave equation is second order in time, just like Newton's second law, so initial displacement and velocity must be specified at each point to find a unique solution.

To numerically solve the classical wave equation via staggered leapfrog we approximate both the temporal and spatial derivatives in Eq. (5.1) with centered finite differences, like this:

$$\begin{aligned}\frac{\partial^2 y}{\partial t^2} &\approx \frac{y_j^{n+1} - 2y_j^n + y_j^{n-1}}{\tau^2} \\ \frac{\partial^2 y}{\partial x^2} &\approx \frac{y_{j+1}^n - 2y_j^n + y_{j-1}^n}{h^2}\end{aligned} \tag{5.5}$$

In this notation, spatial position is indicated by a subscript j, referring to grid points x_j, while position in time is indicated by superscripts n, referring to time points t_n so that $y(x_j, t_n) = y_j^n$. The time steps and the grid spacings are assumed to be uniform with time step called τ and grid spacing called h. The staggered leapfrog algorithm aims to find y_j one time step into the future, denoted by y_j^{n+1}, from the current and previous values of y_j. To derive the algorithm put Eqs. (5.5) into Eq. (5.1) and solve for y_j^{n+1} to find

$$y_j^{n+1} = 2y_j^n - y_j^{n-1} + \frac{c^2\tau^2}{h^2}\left(y_{j+1}^n - 2y_j^n + y_{j-1}^n\right) \tag{5.6}$$

P5.1 Derive Eq. (5.6) using the approximate second derivative formulas. This is really simple, so just do it on paper.

Equation (5.6) can only be used at interior spatial grid points because the $j+1$ or $j-1$ indices reach beyond the grid at the first and last grid points. The behavior of the solution at these two end points is determined by the boundary conditions. Since we will want to use both fixed value and derivative boundary conditions, let's use a cell-centered grid with ghost points (with N cells and $N+2$ grid points) so we can easily handle both types without changing our grid. If the values at the ends are specified we have

$$\begin{aligned}\frac{y_0^{n+1} + y_1^{n+1}}{2} = f_{\text{left}}(t_{n+1}) \quad &\Rightarrow \quad y_0^{n+1} = -y_1^{n+1} + 2f_{\text{left}}(t_{n+1}) \\ \frac{y_{N+1}^{n+1} + y_N^{n+1}}{2} = f_{\text{right}}(t_{n+1}) \quad &\Rightarrow \quad y_{N+1}^{n+1} = -y_N^{n+1} + 2f_{\text{right}}(t_{n+1})\end{aligned} \tag{5.7}$$

If the derivatives are specified then we have

$$\begin{aligned}\frac{y_1^{n+1} - y_0^{n+1}}{h} = g_{\text{left}}(t_{n+1}) \quad &\Rightarrow \quad y_0^{n+1} = y_1^{n+1} - hg_{\text{left}}(t_{n+1}) \\ \frac{y_{N+1}^{n+1} - y_N^{n+1}}{h} = g_{\text{right}}(t_{n+1}) \quad &\Rightarrow \quad y_{N+1}^{n+1} = y_N^{n+1} + hg_{\text{right}}(t_{n+1})\end{aligned} \tag{5.8}$$

To use staggered leapfrog, we first advance the solution at all interior points to the next time step using Eq. (5.6), then we apply the boundary conditions using

the appropriate equation from Eqs. (5.7)-(5.8) to find the values of y at the end points, and then we are ready to take another time step.

The staggered leapfrog algorithm in Eq. (5.6) requires not just y at the current time level y_j^n but also y at the previous time level y_j^{n-1}. This means that we'll need to keep track of three arrays: an array y for the current values y_j^n, an array `yold` for the values at the previous time step y_j^{n-1}, and an array `ynew` for the values at the next time step y_j^{n+1}. At time $t = 0$ when the calculation starts, the initial position condition gives us the current values y_j^n, but we'll have to make creative use of the initial velocity condition to create an appropriate `yold` to get started. To see how this works, let's denote the initial values of y on the grid by y_j^0, the values after the first time step by y_j^1, and the unknown previous values (`yold`) by y_j^{-1}. A centered time derivative at $t = 0$ turns the initial velocity condition from Eq. (5.4) into

$$\frac{y_j^1 - y_j^{-1}}{2\tau} = v_0(x_j) \tag{5.9}$$

This gives us an equation for the previous values y_j^{-1}, but it is in terms of the still unknown future values y_j^1. However, we can use Eq. (5.6) to obtain another relation between y_j^1 and y_j^{-1}. Leapfrog at the first step ($n = 0$) says that

$$y_j^1 = 2y_j^0 - y_j^{-1} + \frac{c^2\tau^2}{h^2}\left(y_{j+1}^0 - 2y_j^0 + y_{j-1}^0\right) \tag{5.10}$$

If we insert this expression for y_j^1 into Eq. (5.9), we can solve for y_j^{-1} in terms of known quantities:

$$y_j^{-1} = y_j^0 - v_0(x_j)\tau + \frac{c^2\tau^2}{2h^2}\left(y_{j+1}^0 - 2y_j^0 + y_{j-1}^0\right) \tag{5.11}$$

P5.2 Derive Eq. (5.11) from Eqs. (5.9) and (5.10). Again, just use paper and pencil.

OK; we are now ready to code the staggered leapfrog algorithm.

P5.3 (a) Start by making a cell-centered x grid with ghost points over the region $0 \le x \le L$, with $L = 1$ m. Use $N = 200$ cells, so you have 202 grid points. Define the initial displacement y as Gaussian bump with 1 cm amplitude in the middle of the string, like this

```
y = 0.01 * np.exp(-(x-L/2)**2 / 0.02)
```

and the initial velocity vy to be zero everywhere along the string. Used fixed-value boundary conditions, with $y(0) = 0$ and $y(L) = 0$. Plot the initial position just to make sure your grid is right and that your initial position array looks reasonable

(b) Assume that the wave speed on this string is $c = 2$ m/s, and pick the time step as `tau = 0.2*h/c`. (We'll explore this choice more later.) Then create a variable `yold` using Eq. (5.11), and enforce the

boundary conditions on `yold`. As you write this code, don't write a bunch of `for` loops to iterate through all the points. Instead, assign all interior points at once using the NumPy array colon indexing method (e.g. `y[1:-1]` accesses all the interior points of `y`) and then set the boundary conditions explicitly.

(c) Now it is time to code the main staggered leapfrog algorithm and make an animation of the solution. Since it has been a couple of labs since we made an animation, here is a framework for the code to remind you of the basic animation commands:

```
ynew = np.zeros_like(y)
j = 0
t = 0
tmax = 2
plt.figure(1) # Open the figure window

# the loop that steps the solution along
while t < tmax:
    j = j+1
    t = t + tau

    # Use leapfrog and the boundary conditions to load
    # ynew with y at the next time step using y and yold

    # update yold and y for next timestep
    # remember to use np.copy

    # make plots every 50 time steps
    if j % 50 == 0:
        plt.clf()          # clear the figure window
        plt.plot(x,y,'b-')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.title('time={:1.3f}'.format(t))
        plt.ylim([-0.03,0.03])
        plt.xlim([0,1])
        plt.draw()         # Draw the plot
        plt.pause(0.1)     # Give the computer time to draw
```

The actual staggered leapfrog code is missing above. You'll need to write that. Run the animations long enough that you can see the reflection from the ends and the way the two pulses add together and pass right through each other.

(d) Once you have it running, experiment with various time steps τ. Show by numerical experimentation that if $\tau > h/c$ the algorithm blows up spectacularly. This failure is called a *numerical instability* and we will be trying to avoid it all semester. This limit is called the *Courant-Friedrichs-Lewy condition*, or sometimes the *CFL condition*, or sometimes (unfairly) just the *Courant condition*.

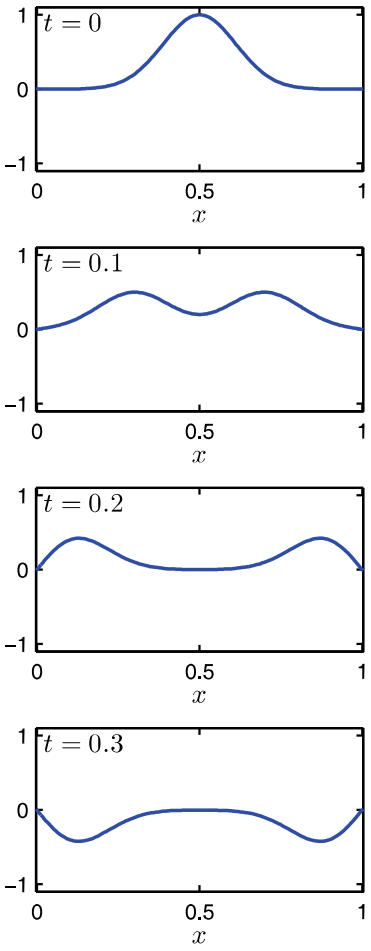

Figure 5.1 Snapshots of the evolution of a wave on a string with fixed ends and an initial displacement but no initial velocity. (See Problem 5.3(b))

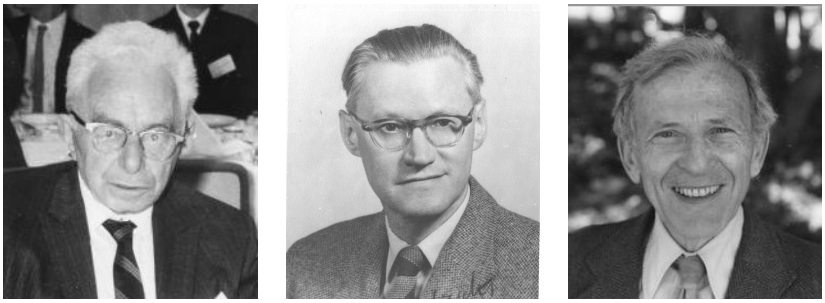

Figure 5.2 Richard Courant (left), Kurt Friedrichs (center), and Hans Lewy (right) described the CFL instability condition in 1928.

(e) Change the boundary conditions so that $\frac{\partial y}{\partial x} = 0$ at each end and watch how the reflection occurs in this case.

(f) Change the initial conditions from initial displacement with zero velocity to initial velocity with zero displacement. Use an initial Gaussian velocity pulse just like the displacement pulse you used earlier and use fixed-end boundary conditions. Watch how the wave motion develops in this case. (You will need to change the y-limits in the `axis` command to see the vibrations with these parameters.) Then find a slinky, stretch it out, and whack it in the middle to verify that the math does the physics right.

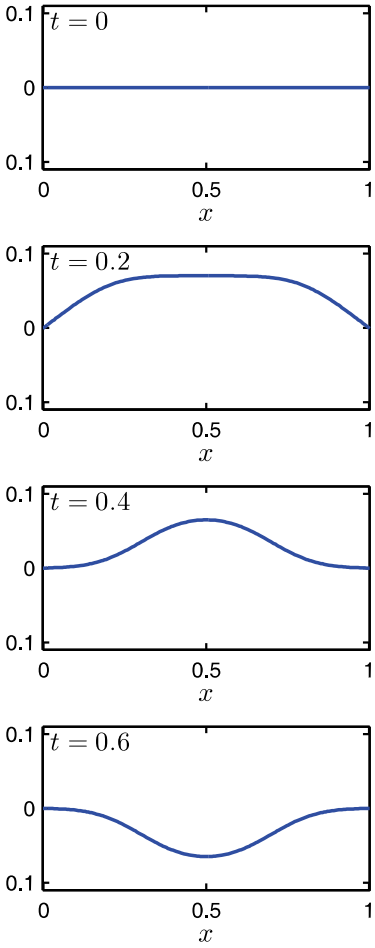

Figure 5.3 Snapshots of the evolution of a wave on a string with fixed ends and no initial displacement but with an initial velocity. (See Problem 5.3(f))

The damped wave equation

We can modify the wave equation to include damping of the waves using a linear damping term, like this:

$$\frac{\partial^2 y}{\partial t^2} + \gamma \frac{\partial y}{\partial t} - c^2 \frac{\partial^2 y}{\partial x^2} = 0 \tag{5.12}$$

with c constant. The staggered leapfrog method can be used to solve Eq. (5.12) also. To do this, we use the approximate first derivative formula

$$\frac{\partial y}{\partial t} \approx \frac{y_j^{n+1} - y_j^{n-1}}{2\tau} \tag{5.13}$$

along with the second derivative formulas in Eqs. (5.5) and find an expression for the values one step in the future:

$$y_j^{n+1} = \frac{1}{2+\gamma\tau}\left(4y_j^n - 2y_j^{n-1} + \gamma\tau y_j^{n-1} + \frac{2c^2\tau^2}{h^2}\left(y_{j+1}^n - 2y_j^n + y_{j-1}^n\right)\right) \tag{5.14}$$

P5.4 (a) Derive Eq. (5.14).

(b) Find a new formula for the initial value of `yold` using Eqs. (5.9) and (5.14). When you get the answer, ask your TA or instructor to check to see if you got it right.

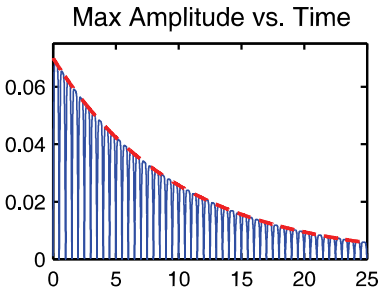

Figure 5.4 The maximum amplitude of oscillation decays exponentially for the damped wave equation. (Problem 5.4(c))

(c) Modify your staggered leapfrog code to include damping with $\gamma = 0.2$. Then run your animation with the initial conditions in Problem 5.3(f) and verify that the waves damp away. You will need to run for about 25 s and you will want to use a big skip factor so that you don't have to wait forever for the run to finish. Include some code to record the maximum value of $y(x)$ over the entire grid as a function of time and then plot it as a function of time at the end of the run so that you can see the decay caused by γ. The decay of a simple harmonic oscillator is exponential, with amplitude proportional to $e^{-\gamma t/2}$. Scale this time decay function properly and lay it over your maximum y plot to see if it fits. Can you explain why the fit is as good as it is? (Hint: think about doing this problem via separation of variables.)

The damped and driven wave equation

Finally, let's look at what happens when we add an oscillating driving force to our string, so that the wave equation becomes

$$\frac{\partial^2 y}{\partial t^2} + \gamma \frac{\partial y}{\partial t} - c^2 \frac{\partial^2 y}{\partial x^2} = \frac{f(x)}{\mu} \cos(\omega t) \tag{5.15}$$

At the beginning of Lab 3 we discussed the qualitative behavior of this system. Recall that if we have a string initially at rest and then we start to push and pull on a string with an oscillating force/length of $f(x)$, we launch waves down the string. These waves reflect back and forth on the string as the driving force continues to launch more waves. The string motion is messy at first, but the damping in the system causes the the transient waves from the initial launch and subsequent reflections to eventually die away. In the end, we are left with a steady-state oscillation of the string at the driving frequency ω.

Now that we have the computational tools to model the time evolution of the system, let's watch this behavior.

P5.5 (a) Re-derive the staggered leapfrog algorithm to include both driving and damping forces as in Eq. (5.15).

(b) Modify your code from Problem 5.4 to use this new algorithm. We'll have the string start from rest, so you don't need to worry about finding y_{old}. Just set $y = 0$ and $y_{\text{old}} = 0$ and enter the time-stepping loop.

This problem involves the physics of waves on a real guitar string, so we'll need to use realistic values for our parameters. Use $T = 127$, $\mu = 0.003$, and $L = 1.2$ (in SI units) and remember that $c = \sqrt{T/\mu}$. Use the same driving force as in Problem 3.2(a)

$$f(x) = \begin{cases} 0.73 & \text{if } 0.8 \le x \le 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.16}$$

and set the driving frequency at $\omega = 400$. Choose a damping constant γ that is the proper size to make the system settle down to steady state after 20 or 30 bounces of the string. (You will have to think about the value of ω that you are using and about your damping rate result from problem 5.4 to decide which value of γ to use to make this happen.)

Run the model long enough that you can see the transients die away and the string settle into the steady oscillation at the driving frequency. You may find yourself looking at a flat-line plot with no oscillation at all. If this happens look at the vertical scale of your plot and remember that we are doing real physics here. If your vertical scale goes from -1 to 1, you are expecting an oscillation amplitude of 1 meter on your guitar string. Compare the steady state mode to the shape found in Problem 3.2(a) (see Fig. 3.1).

Then run again with $\omega = 1080$, which is close to a resonance, and again see the system come into steady oscillation at the driving frequency.

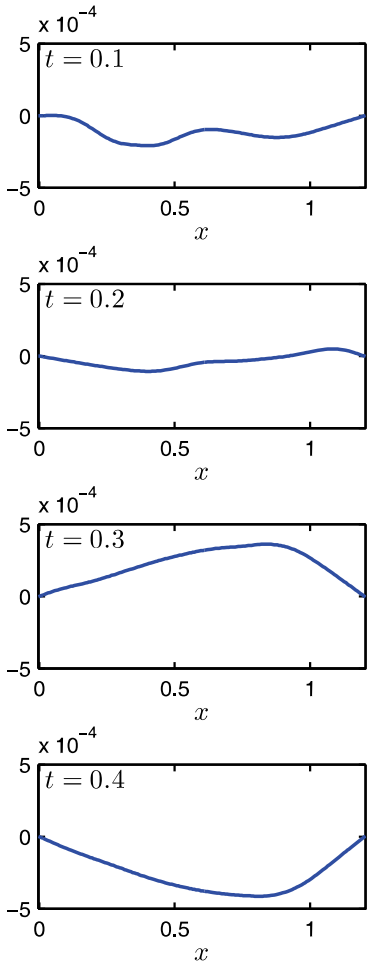

Figure 5.5 Snapshots of the evolution a driven and damped wave with $\omega = 400$. As the transient behavior dies out, the oscillation goes to the resonant mode. To make the pictures more interesting, the string was not started from rest in these plots. (In Problem 5.5 you start from rest for easier coding.)