Chapter 21

Partial Differential Equations I

Partial differential equations (PDEs) describe systems with more than one independent variable. Many examples involve wave motion, such as water waves, electromagnetic waves, sound waves, *etc.* In each of these cases, the dependent variable is the amplitude of the wave and the independent variables are the time and space coordinates. Other examples of PDEs include the Schrödinger equation, which describes the quantum mechanical wave function, the Poisson equation, which describes the electrostatic and gravitational potentials, and the heat equation, which describes the flow of thermal energy through a body.

21.1 Waves on a string

Consider a string stretched along the x–axis, under tension T. Perhaps a guitar string or a violin string. The mass per unit length of the string is μ. We will ignore gravity and consider only small amplitude, transverse waves. That is, we only allow small displacements of the string away from the x axis in the y direction. Our goal is to describe the displacement amplitude $y(t,x)$ as a function of the two independent variables t and x.

Figure 21.1 shows a short segment of string. The forces on the right and left ends of the segment depend on the tension T and the angles θ_R, θ_L that the ends make with the x direction. The y component of force on the string segment is

$$F_y = T\sin\theta_R - T\sin\theta_L \ . \tag{21.1}$$

By applying Newton's second law we find

$$\mu\Delta x \frac{d^2y}{dt^2} = T(\sin\theta_R - \sin\theta_L) \ , \tag{21.2}$$

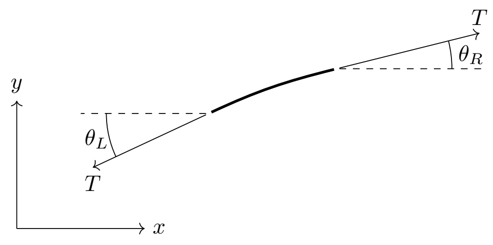

Fig. 21.1: A segment of string under tension T.

where Δx is the length of the segment (assuming displacements are small) and $\mu\Delta x$ is the segment's mass. For small displacements of the string the angles θ_R and θ_L will be small. To a good approximation the sin's can be replaced with tan's. But the tangent of θ_R is the slope dy/dx at the right end of the segment. Likewise, the tangent of θ_L is the slope at the left end of the segment. The equation of motion becomes

$$\mu \frac{d^2y}{dt^2} = \frac{T}{\Delta x}\left(\left.\frac{dy}{dx}\right|_R - \left.\frac{dy}{dx}\right|_L \right) \ . \tag{21.3}$$

In the limit as the segment size shrinks to zero, the right–hand side (excluding the factor T) equals the second derivative of y; that is,

$$\lim_{\Delta x\to 0} \frac{1}{\Delta x}\left(\left.\frac{dy}{dx}\right|_R - \left.\frac{dy}{dx}\right|_L \right) = \frac{d^2y}{dx^2} \ . \tag{21.4}$$

Therefore the PDE that governs small–amplitude wave motion on a string is

$$\frac{d^2y}{dt^2} = c^2 \frac{d^2y}{dx^2} \ , \tag{21.5}$$

where $c^2 = T/\mu$. This is the *wave equation.* Using dots to denote derivatives with respect to t and primes to denote derivatives with respect to x, the wave equation can be written as $\ddot{y} = c^2 y''$.

It is easy to verify that the wave equation has solutions of the form

$$y(t,x) = F(x+ct) + G(x-ct) \ , \tag{21.6}$$

where F and G are arbitrary functions of their arguments. In fact, Eq. (21.6) is the general solution. Any solution of the wave equation can be described as a superposition of a left–moving wave $F(x+ct)$ and a right–moving wave $G(x-ct)$. A purely left–moving wave $y(t,x) = F(x+ct)$ satifies $y(t+T, x-cT) = y(t,x)$; the waveform shifts to the left by an amount

cT during time T. Likewise, a purely right–moving wave $y(t,x) = G(x-ct)$ satisfies $y(t+T, x+cT) = y(t,x)$; the waveform shifts to the right by an amount cT during time T. The constant c is the *wave speed*.

Exercise 21.1

Verify that $y(t,x) = F(x+ct)+G(x-ct)$ satisfies the wave equation for any functions F and G.

21.2 CTCS algorithm

A PDE such as (21.5) can be solved numerically through finite differencing. Let us assume that the ends of the string are held fixed at locations x_a, x_b on the x–axis. Divide the spatial interval $x_a \leq x \leq x_b$ into J segments, each of length $\Delta x = (x_b - x_a)/J$. This defines $J+1$ *spatial nodes* at locations $x_0, x_1, \ldots, x_J$. That is, $x_j = x_a + j\Delta x$ where $j = 0, \ldots, J$. We also divide the time axis into timesteps of length Δt. The *temporal nodes* are denoted $t^0, t^1, \ldots, t^N$. That is, $t^n = n\Delta t$ where $n = 0, \ldots, N$. Let[1]

$$y_j^n \equiv y(t^n, x_j) \tag{21.7}$$

denote the amplitude of the string at spatial location x_j and time t^n. See Figure 21.2.

The derivatives of $y(t,x)$, evaluated at node x_j and t^n, can be approximated as

$$\ddot{y}(t^n, x_j) \approx \frac{y_j^{n+1} - 2y_j^n + y_j^{n-1}}{\Delta t^2} \ , \tag{21.8a}$$

$$y''(t^n, x_j) \approx \frac{y_{j+1}^n - 2y_j^n + y_{j-1}^n}{\Delta x^2} \ , \tag{21.8b}$$

using the three–point centered difference formula (16.21). Inserting these expressions into the PDE (21.5) and solving for y_j^{n+1}, we find

$$y_j^{n+1} = 2y_j^n - y_j^{n-1} + \left(\frac{c\Delta t}{\Delta x}\right)^2 (y_{j+1}^n - 2y_j^n + y_{j-1}^n) \ . \tag{21.9}$$

The numerical method based on this approximation to the wave equation is called the centered–time, centered–space (CTCS) algorithm.

Here's the idea. Equation (21.9) must hold for each value of the time index n. In particular, for $n = 1$, we have

$$y_j^2 = 2y_j^1 - y_j^0 + \left(\frac{c\Delta t}{\Delta x}\right)^2 (y_{j+1}^1 - 2y_j^1 + y_{j-1}^1) \ . \tag{21.10}$$

[1] The superscripts on t and y are indices, not powers.

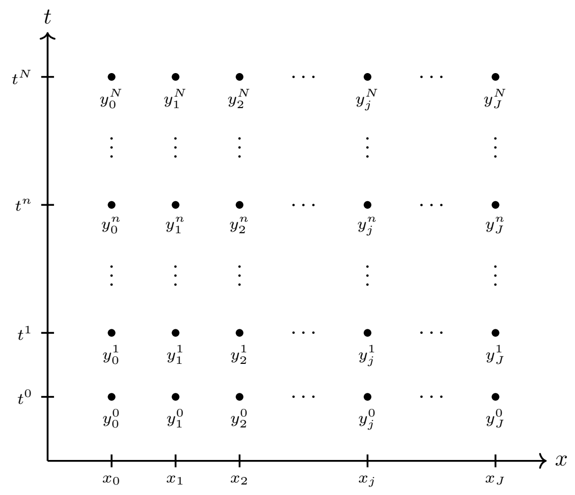

Fig. 21.2: The string lies along the x–axis, which is divided into J segments of length Δx. The $J+1$ spatial nodes are denoted $x_0, \ldots, x_J$, where $x_0 \equiv x_a$ is the left end and $x_J \equiv x_b$ is the right end. The time axis is divided into timesteps of length Δt. The temporal nodes are denoted $t^0, \ldots, t^N$. The amplitude of the string at spatial location x_j and time t^n is denoted y_j^n.

Thus, if we know the amplitudes at each spatial node at times t^0 and t^1, we can use Eq. (21.10) to predict the amplitudes at time t^2 at each of the interior nodes $j = 1, \ldots, J-1$. Since the endpoints of the string are fixed, the amplitudes at the endpoints are simply $y_0^2 = y_0^1$ and $y_J^2 = y_J^1$. Having found all of the amplitudes at time t^2, we can apply Eq. (21.9) with $n = 2$ to obtain the amplitudes at t^3. Similarly we find the amplitudes at t^4, t^5, *etc.*

Let's put together a numerical code to solve the wave equation (21.5) using the CTCS algorithm.

- Import NumPy (as `np`) and other libraries.
- Choose values for the parameters including x_a, x_b and c.
- Set up the numerical grid; for example:

```
J = 100          # number of space intervals
N = 500          # number of timesteps
dx = (xb-xa)/J
```

```
dt = 0.2*dx/c
x = np.linspace(xa,xb,J+1)
```

Here, the timestep `dt` is chosen to be proportional to `dx/c`; we will discuss this later.

- Define arrays for y^{n-1}, y^n and y^{n+1}:

```
ym = np.zeros(J+1)    # ym = "y minus" = y at time t^(n-1)
y = np.zeros(J+1)     # y at time t^n
yp = np.zeros(J+1)    # yp = "y plus"  = y at time t^(n+1)
```

- Choose “initial” data; that is, data at times t^0 and t^1:

```
y = some function of x
yp = some function of x
```

The array `y` holds the values of y^0_j, and the array `yp` holds the values of y^1_j. This completes the first timestep.

- In preparation for the second timestep, from t^1 to t^2, copy the arrays `y` and `yp` into `ym` and `y`:

```
ym = np.copy(y)
y = np.copy(yp)
```

- Evolve the system forward in time using the CTCS scheme (21.9):

```
for n in range(2,N+1):
   for j in range(1,J):
      yp[j] = 2.0*y[j] - ym[j]\
         + (c*dt/dx)**2 *(y[j+1] - 2.0*y[j] + y[j-1])
   yp[0] = y[0]
   yp[J] = y[J]
   ym = np.copy(y)
   y = np.copy(yp)
```

The loop over n begins at $n = 2$ because the first timestep was taken when we assigned values to y^1_j. Inside the loop over n, `yp[j]` is computed for all interior nodes $j = 1, \ldots, J-1$. Boundary conditions are used to determine the endpoint values `yp[0]` and `yp[J]`. The arrays `y` and `yp` are then copied to `ym` and `y` in preparation for the next timestep.

- Add code as necessary to plot graphs and print (or save) `y` at various times t^n. For example, to plot $y(x)$ after every 100 timesteps, simply include

```
if (n%100 == 0):
   matplotlib.pyplot.plot(x,y)
```

inside the loop over n.

Since the wave equation contains second order derivatives in time, the initial data consist of the initial shape $y(0, x)$ as well as the initial velocity

$\dot{y}(0,x)$. For example, let the string start from rest with a "Gaussian pulse" shape

$$y(0,x) = Ae^{-x^2/\sigma^2} \ , \tag{21.11}$$

where A and σ are constants. You can implement this in your code by setting

$$y_j^0 = Ae^{-x_j^2/\sigma^2} \tag{21.12}$$

for all j. Since the string is initially at rest, $\dot{y}(0,x) = 0$. To implement this condition, replace the time derivative with the two–point forward difference approximation (16.4). This yields $(y_j^1 - y_j^0)/\Delta t = 0$, so that

$$y_j^1 = y_j^0 \tag{21.13}$$

for all j. In the numerical code decribed above, y_j^0 is placed into the array `y` and y_j^1 is placed into the array `yp`.

Exercise 21.2a

Write a code to solve the wave equation with boundary conditions $y(t,x_a) = y(t,x_b) = \text{const}$ at the endpoints $x_a = -5.0$ and $x_b = 5.0$ (in SI units). Use the CTCS scheme with $J = 100$ or more. Choose the mass density $\mu = 0.01$ and tension $T = 25.0$, which yields a wave speed of $c = 50.0$. Start the string from rest, Eq. (21.13), with a Gaussian shape, Eq. (21.12). Choose any value for the constant A, such as $A = 1$. Choose a value for σ such that the endpoint values y_0^0 and y_J^0 are close to zero. Have your code plot y versus x at various times on the same graph.

Exercise 21.2b

Estimate the location of the peak in $y(t,x)$ at various times, and use this to verify that the wave speed is $c = 50.0$.

You might wonder why we use the one–dimensional arrays `ym[j]`, `y[j]`, and `yp[j]` to represent y_j^{n-1}, y_j^n, and y_j^{n+1}. Why not create a single two–dimensional array `y[n,j]` to hold the string amplitudes for each spatial node x_j at each time t^n? This would be easier to implement numerically. The difficulty is that the array `y[n,j]` would contain $(N+1)(J+1)$ elements. If N and J are large, your computer might run into memory issues.

This can be especially problematic for PDEs in two or three spatial dimensions. For example, sound waves are governed by the wave equation in three dimensions,

$$\frac{\partial^2 p}{\partial t^2} = c^2 \left[\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} + \frac{\partial^2 p}{\partial z^2} \right] , \tag{21.14}$$

where p is the air pressure. If we build a numerical code with J subintervals in each of the three spatial directions, the complete array of pressures would have $(N+1)(J+1)^3$ elements. With $N = 10000$ and $J = 500$ (not unreasonable numbers), this is more than 1.2×10^{12} elements.

21.3 Numerical instability

Exercise 21.3a

Change the timestep in your code to $\Delta t = 1.2\Delta x/c$. (Use $J = 100$. Keep the remaining parameters and the initial conditions unchanged.) Plot *separate* graphs of $y(x)$ at times $25\Delta t$ and $40\Delta t$.

As a general rule, a finite difference scheme for PDEs will become *numerically unstable* if the timestep Δt is too large. When this happens the numerical results are meaningless—the discrete values y_j^n do not approximate any solution of the original continuum PDE.

For the wave equation, $\Delta x/c$ is the time it takes a wave to travel from one spatial node to the next. The timestep is usually defined as a multiple of $\Delta x/c$, such as

$$\Delta t = \eta \Delta x/c \ . \tag{21.15}$$

Many numerical PDE methods will yield physically meaningful results only if η is small, less than some maximum value $\eta_{\rm max}$. The value of $\eta_{\rm max}$ depends on the details of the PDE and the details of the numerical algorithm, but typically $\eta_{\rm max}$ is of order unity. The restriction $\Delta t \le \eta_{\rm max}\Delta x/c$ is called the *Courant* (or *Courant–Friedrichs–Lewy*) condition.

Exercise 21.3b

Run numerical experiments to find η_{max} (approximately) for the CTCS algorithm.

For some numerical algorithms, not considered here, $\eta_{\rm max}$ is infinite. Such a scheme is *unconditionally stable.* Although the scheme might be stable for arbitrarily large Δt, it is usually not accurate for large Δt. Some numerical algorithms are *unconditionally unstable*, that is, they are unstable for any value of Δt.

Let's take a close look at the instability in the CTCS method. First, we need to recognize that the wave equation (21.5) is a *linear* partial differential equation. If $y(t,x)$ and $\tilde{y}(t,x)$ are two solutions of the wave equation, then $y(t,x)+\epsilon\tilde{y}(t,x)$ is also a solution. Here, ϵ is a constant parameter. The same property holds for the discrete equation (21.9): if both y_j^n and $\tilde{y}_j^n$ satisfy the discrete wave equation, then $y_j^n+\epsilon\tilde{y}_j^n$ is also a solution.

Let y_j^n denote the physical solution of the discrete wave equation that we hope to find from our numerical code. An instability arises when the discrete wave equation also has an unphysical solution $\tilde{y}_j^n$ that grows without bound. The unphysical solution can become superimposed on top of the physical solution, so what the code actually finds is $y_j^n+\epsilon\tilde{y}_j^n$. Even if ϵ is very small, the amplitude of the unphysical term $\epsilon\tilde{y}_j^n$ can grow without bound and eventually overtake the physical solution.

The physical part of the solution, y_j^n, is determined by our choice of initial conditions. The unphysical part $\epsilon\tilde{y}_j^n$ creeps into the simulation due to machine roundoff errors. If the amplitude for the physical solution is of order unity (that is, typical values for y_j^n are around 1, to within a factor of ten or so) then the *initial* amplitude of the unphysical part $\epsilon\tilde{y}_j^n$ will be the size of machine roundoff error, roughly 10^{-16}. If we scale the unphysical solution $\tilde{y}_j^n$ so that its initial amplitude is of order unity, then $\epsilon\approx 10^{-16}$.

Figure 21.3 shows the results of a simulation of the wave equation using the CTCS method, with $\Delta t = 1.2\Delta x/c$. This violates the Courant condition. The initial data consist of a Gaussian pulse (21.11), initially at rest, with $A=1$ and $\sigma=1$. The pulse quickly splits into left–moving and right–moving pieces, each with amplitude 0.5. Superimposed on the physical solution (smooth, double pulse) is an unphysical part (the jagged spikes). The unphysical part of the solution shown in Fig. 21.3 grows very rapidly. At the time shown, $t=0.072$, the amplitude of the spikes is ≈ 0.6. By the time $t=0.1$, the spikes have grown to $\approx 10^6$. Even though the unphysical part of the solution begins with a small amplitude dictated by machine roundoff errors, it grows so quickly that the simulation is spoiled after only 30 timesteps.

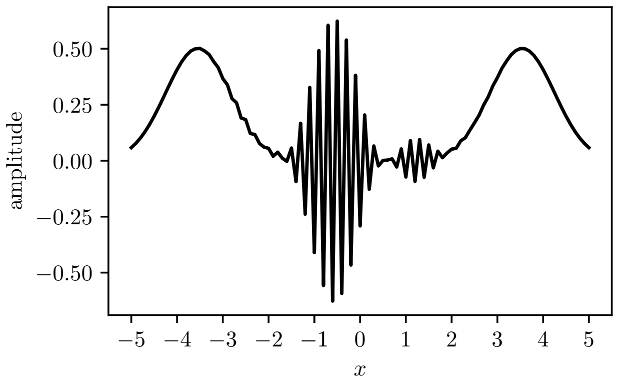

Fig. 21.3: Simulation of a Gaussian wave pulse using the CTCS method with Δt beyond the Courant limit. The figure shows the wave at time $t = 0.072$, after $n = 30$ timesteps.

Exercise 21.3c

Reproduce the simulation from Fig. 21.3 by setting (in SI units) $A = 1$, $\sigma = 1$, $c = 50$, $x_a = -10$, $x_b = 10$, and $J = 200$. Set the timestep to $\Delta t = 1.2\Delta x/c$. Use the NumPy function `np.max(y)` to determine the maximum amplitude at the end of each timestep. Plot the natural log of the maximum amplitude versus the timestep number n for $0 \leq n \leq 50$.

For $0 \leq n < 30$, the simulation is dominated by the physical solution so the maximum amplitude is of order unity. For $n > 30$, the simulation is dominated by the unphysical solution and the maximum amplitude grows exponentially.

Let $\epsilon\tilde{y}_{max}(n)$ denote the maximum amplitude of the unphysical part of the solution, as a function of the timestep n. Since the unphysical solution grows exponentially, we have $\epsilon\tilde{y}_{max}(n) = Ke^{sn}$ for some constants K and s. If the unphysical solution is "seeded" by machine roundoff errors in the early timesteps, we would expect that for very small n, $\epsilon\tilde{y}_{max}(n)$ should be smaller than the physical signal by a factor of roughly 10^{-16}.

Exercise 21.3d

Determine the constants K and s in the formula $\epsilon\tilde{y}_{max}(n) = Ke^{sn}$, using your data for $n \geq 32$. Use this result to compute $\epsilon\tilde{y}_{max}(0)$ and $\epsilon\tilde{y}_{max}(1)$. Are these on the order of machine error?

This calculation should help convince you that the amplitude of the unphysical part of the solution can grow from the size of machine error to order unity in just 30 or so timesteps. Instabilites don't always grow this quickly. Sometimes an unstable solution can take 1000's or more timesteps to overtake the physical solution and spoil the simulation.

21.4 An unphysical solution

When the Courant condition $\Delta t \leq \eta_{max}\Delta x/c$ is violated, the discrete wave equation (21.9) admits many unphysical solutions. One such solution is fairly simple to write down:

$$\tilde{y}^n_j = \mathrm{Re}\Big[\big(1 - 2\eta^2 - 2\eta\sqrt{\eta^2 - 1}\big)^n\Big](-1)^j \; . \tag{21.16}$$

The symbol Re stands for the real part of the expression that follows. Recall from Eq. (21.15) that $\eta = c\Delta t/\Delta x$.

There are two cases to consider. If $\eta \geq 1$, then the square root term $\sqrt{\eta^2 - 1}$ is real. The entire expression in square brackets is real and the Re symbol can be dropped. On the other hand, if $\eta < 1$, the square root is imaginary: $\sqrt{\eta^2 - 1} = i\sqrt{1 - \eta^2}$. In this case Re instructs us to take the real part of the complex expression in square brackets.

Exercise 21.4a

Verify that Eq. (21.16) is a solution of the discrete wave equation (21.9). Hint: it suffices to show that

$$\tilde{Y}^n_j = \big(1 - 2\eta^2 - 2\eta\sqrt{\eta^2 - 1}\big)^n(-1)^j$$

is a solution for any η. Since the wave equation is linear with real coefficients, the complex conjugate $(\tilde{Y}^n_j)^*$ must also be a solution. It follows that the real part $\tilde{y}^n_j = \mathrm{Re}\,\tilde{Y}^n_j = [\tilde{Y}^n_j + (\tilde{Y}^n_j)^*]/2$ is a solution.

The factor $(-1)^j$ in Eq. (21.16) is a clear indication that the solution is unphysical. The amplitude $\tilde{y}^n_j$ switches from positive to negative from one spatial node to the next, creating a "zig–zag" appearance in a plot of y versus x. This behavior depends crutially on our choices for the location and spacing of the spatial nodes x_j. Of course, a physical solution can have closely spaced peaks and valleys as well, but the distance between peaks and valleys should not be determined by the computer programmer's choice of node spacing.

Exercise 21.4b

Set the initial data in a CTCS code to match the unphysical solution:

$$\tilde{y}_j^0 = (-1)^j \ ,$$

$$\tilde{y}_j^1 = \begin{cases} \left[1 - 2\eta^2\right](-1)^j \ , & \text{if } 0 < \eta \le 1 \\ \left[1 - 2\eta^2 - 2\eta\sqrt{\eta^2 - 1}\right](-1)^j \ , & \text{if } \eta > 1 \ . \end{cases} ,$$

Plot $\tilde{y}_j^n$ versus j for various values of n, for both cases $0 < \eta \le 1$ and $\eta > 1$. Also plot $\tilde{y}_j^n$ versus n with a fixed value of j, for both $0 < \eta \le 1$ and $\eta > 1$.