Chapter 20

Boundary Value Problems

20.1 The hanging cord

What is the shape of a hanging cord? We can make this question precise. Suspend a cord (or rope or chain) of length L from its ends. The ends are fixed to the points (x_a, y_a) and (x_b, y_b) in the x–y plane, where x is horizontal and y is vertical. What is the curve $y(x)$ taken by the cord?

To answer this question, we first derive an ordinary differential equation satisfied by the function $y(x)$. The equation can be solved numerically as a *boundary value problem*. With a boundary value problem, the freely chosen data are specified by *boundary conditions*. Boundary conditions are specified at the boundaries (or endpoints) of the system. For the hanging cord, the boundary data consist of the numerical values y_a and y_b at the endpoints x_a and x_b.

In contrast, for an *initial value problem* the freely chosen data are given at the initial time. In the previous chapters on differential equations we only considered initial value problems. For dynamical systems such as the driven, damped pendulum of Ch. 18, the initial data are typically the initial positions and initial velocities.

20.2 Differential equation for the cord

Divide the cord into segments of length $\Delta\ell$. Figure 20.1 shows one of these segments. The forces $\vec{F}_L$ and $\vec{F}_R$ on the segment come from the tension in the cord. The force $\vec{F}_L$ acts on the left (L) end of the segment, and the force $\vec{F}_R$ acts on the right (R) end of the segment. There is also a downward force due to gravity, $\Delta m g$, where Δm is the mass of the segment.

Since the segment of cord is in static equilibrium, the sum of forces must

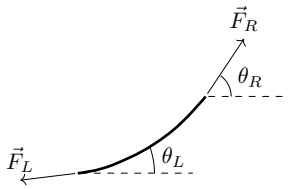

Fig. 20.1: Each segment of cord is acted upon by three forces; tension on the right, tension on the left, and the downward gravitational force (not shown).

vanish. Thus,

$$\vec{F}_L + \vec{F}_R - \Delta m g \hat{y} = 0 \ , \tag{20.1}$$

where $\hat{y}$ is the unit vector pointing upward. The force on the right end of the segment can be split into x (horizontal) and y (vertical) components:

$$\vec{F}_R = T_R \cos\theta_R \, \hat{x} + T_R \sin\theta_R \, \hat{y} \ . \tag{20.2}$$

Here, T_R is the tenson in the cord at the right end of the segment and θ_R is the angle between the cord and the $\hat{x}$ direction at the right end of the segment. Likewise, for the left end of the segment,

$$\vec{F}_L = -T_L \cos\theta_L \, \hat{x} - T_L \sin\theta_L \, \hat{y} \tag{20.3}$$

Putting these results together we see that the vector equation (20.1) yields

$$T_R \cos\theta_R - T_L \cos\theta_L = 0 \ , \tag{20.4a}$$

$$T_R \sin\theta_R - T_L \sin\theta_L = \Delta m g \ , \tag{20.4b}$$

for the x and y components.

Consider the first of these results. Since the left and right ends of the segment can be chosen as any two points along the cord, this equation tells us that the product $T\cos\theta$ is constant throughout the cord. Although the tension T and angle θ can vary along the cord, we have

$$T\cos\theta = \tau_x \ , \tag{20.5}$$

where τ_x is a constant. This constant τ_x is the x–component of the force due to tension.

Now consider the second equation. This can be rewritten as

$$T_R \cos\theta_R \, \tan\theta_R - T_L \cos\theta_L \, \tan\theta_L = \Delta m g \ . \tag{20.6}$$

Since $T\cos\theta$ equals τ_x for both the left and right ends of the segment, we have

$$\tan\theta_R - \tan\theta_L = \Delta m g/\tau_x \ . \tag{20.7}$$

This can be written more compactly as

$$\Delta(\tan\theta) = \Delta m g/\tau_x \ , \tag{20.8}$$

where $\Delta(\tan\theta) = \tan\theta_R - \tan\theta_L$ is the change in $\tan\theta$ across the segment.

Let μ denote the mass per unit length of the cord. The mass of the segment is $\Delta m = \mu\Delta\ell$, where $\Delta\ell$ is the segment's length. In the limit as $\Delta\ell$ goes to zero, we can approximate the segment as a straight line and the Pythagorean theorem gives

$$\Delta\ell = \sqrt{\Delta x^2 + \Delta y^2} = \Delta x\sqrt{1 + (\Delta y/\Delta x)^2} \ . \tag{20.9}$$

Inserting these results into Eq. (20.8) yields

$$\frac{\Delta(\tan\theta)}{\Delta x} = (\mu g/\tau_x)\sqrt{1 + (\Delta y/\Delta x)^2} \ . \tag{20.10}$$

In the limit $\Delta\ell \to 0$, $\Delta y/\Delta x$ is just the derivative dy/dx. Likewise, $\Delta(\tan\theta)/\Delta x$ is the derivative $d(\tan\theta)/dx$. Recall that at any point on the curve, $\tan\theta$ is the slope at that point: $\tan\theta = dy/dx$. Therefore $d(\tan\theta)/dx = d^2y/dx^2$ and Eq. (20.10) becomes

$$\frac{d^2y}{dx^2} = (\mu g/\tau_x)\sqrt{1 + (dy/dx)^2} \ . \tag{20.11}$$

This ordinary differential equation describes the shape of a hanging cord.

The general solution of Eq. (20.11) is

$$y(x) = a + \frac{\cosh((\mu g/\tau_x)x + b)}{(\mu g/\tau_x)} \ . \tag{20.12}$$

This curve is called a *catenary*. The constants a and b are determined by the boundary conditions.

Exercise 20.2

Verify that the catenary (20.12) satisfies the differential equation (20.11). Then let $g = 9.8\,\mathrm{m/s^2}$, $\mu = 0.1\,\mathrm{kg/m}$, $\tau_x = 0.3\,\mathrm{kg\cdot m/s^2}$ and use the boundary conditions $y(0) = 0.0\,\mathrm{m}$, $y(1) = 0.5\,\mathrm{m}$ to determine the constants a and b. Plot $y(x)$ versus x. (Suggestion: Use SymPy.)

20.3 Relaxation

Equation (20.11) defines a boundary value problem for the shape of a hanging cord. The boundary values are the values of y at the endpoints x_a and x_b. A simple numerical technique for solving such a problem is called *relaxation*. Divide the x–axis into N equal subintervals, each of width Δx, as shown in Fig. 20.2. The x–coordinate values of the nodes are x_i where $i = 0, \ldots N$. The endpoints are $x_0 = x_a$ and $x_N = x_b$. The y coordinate value of the cord at node x_i is denoted y_i.

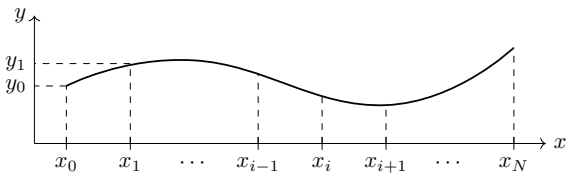

Fig. 20.2: The discrete grid has nodes x_i for $i = 0, \ldots N$. The endpoints are $x_0 = x_a$ and $x_N = x_b$. The separation between adjacent nodes is Δx.

We begin by discretizing the catenary equation Eq. (20.11). In particular, we need discrete approximations for the derivatives $y' = dy/dx$ and $y'' = d^2y/dx^2$. The 3–point centered difference approximations for the first and second derivatives at node x_i are

$$y'(x_i) \approx \frac{y_{i+1} - y_{i-1}}{2\Delta x} \ , \tag{20.13a}$$

$$y''(x_i) \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta x^2} \ . \tag{20.13b}$$

These are Eqs. (16.18) and (16.21) with a few changes in notation. Inserting these expressions into the catenary equation, we have

$$(y_{i+1} - 2y_i + y_{i-1})/\Delta x^2 = (\mu g/\tau_x)\sqrt{1 + (y_{i+1} - y_{i-1})^2/(2\Delta x)^2} \tag{20.14}$$

Now solve for y_i:

$$y_i = \frac{1}{2}(y_{i+1} + y_{i-1}) - (\mu g/\tau_x)\frac{\Delta x^2}{2}\sqrt{1 + (y_{i+1} - y_{i-1})^2/(2\Delta x)^2} \ . \tag{20.15}$$

What can we do with this result? It appears useless. After all, we can't use it to find y_i since we don't know y_{i+1} or y_{i-1}. However, if we have a reasonable approximation for each of the y_i's, we can use this relationship to improve that approximation.

To implement this idea, start with a trial solution. A trial solution is a guess for each value of y_i that satisfies the boundary conditions $y_0 = y_a$ and $y_N = y_b$. We then apply Eq. (20.15) to each of the interior nodes $i = 1, \ldots, N-1$, one after the other. That is, we evaluate the right–hand side of Eq. (20.15) at $i = 1$ and use the result as a new value for y_1. Then evaluate the right–hand side of Eq. (20.15) at $i = 2$ and use the result as a new value for y_2. Continue for each value of i through $N-1$.

This calculation is called a *relaxation sweep*. The end result of a single relaxation sweep is an improved set of values for $y_1, \ldots, y_{N-1}$. (The values y_0 and y_N do not need to be improved, since they are fixed by the boundary conditions.) We now execute a second relaxation sweep by inserting the improved y_i's into the right–hand side of Eq. (20.15). This process is repeated, as many times as needed, until the values for y_i *relax* to an approximate solution of Eq. (20.14).

The relaxation process is typically slow, and the exact solution of Eq. (20.14) is never actually reached. How many relaxation sweeps should you use? In practice, your code should continue until the discrete equation (20.14) is satisfied to some desired degree of accuracy. We will addess this issue in Sec. 20.6.

Keep in mind: Even if we could find the exact solution of the discrete equation (20.14), it would not be an exact solution of the original differential equation (20.11). This is due to truncation errors. The discrete equation is obtained from the differential equation by replacing derivatives with centered difference formulas. These formulas are approximations, with errors proportional to Δx^2.

Exercise 20.3

Write a code to solve the differential equation (20.11). Set $(\mu g/\tau_x) = 5.0$ and choose boundary conditions $y(0) = 0$ and $y(1) = 2$. Use 5000 relaxation sweeps and $N = 101$ nodes. For your trial solution, use the straight line $y_i = 2.0\, x_i$. Have your code plot y versus x after every 1000 sweeps. How much does the value of y at the midpoint, `y[51]`, change between 1000 and 2000 sweeps? Between 2000 and 3000 sweeps? 3000 and 4000? 4000 and 5000?

20.4 Gauss–Seidel and Jacobi methods

You probably wrote your code using a `for` loop, or similar control structure, to sweep across the interior nodes $i = 1, \ldots, N-1$. For example, you might use

```
for i in range(1,N):
     y[i] = 0.5*(y[i+1] + y[i-1]) + ...
```

Let's unwind the `for` loop. The first time through the loop, $i = 1$ and the computer executes the command `y[1] = 0.5*(y[2] + y[0]) + ...`. The right–hand side is evaluated using the trial values for `y[2]` and `y[0]`, then the computer overwrites the trial value for `y[1]` with the right–hand side value. The second time through the loop, $i = 2$ and the computer executes the command `y[2] = 0.5*(y[3] + y[1]) + ...`. The right–hand side is evaluated using the trial value for `y[3]`, and the recently computed new value for `y[1]`. The computer then overwrites the trial value for `y[2]` using the right–hand side value, which is obtained from a mixture of old `y[3]` and new `y[1]`. This mixture of old and new values continues for the remaining cycles through the loop.

Is this really what we want our code to do? An alternative to the code segment above is

```
yold = np.copy(y)
for i in range(1,N):
     y[i] = 0.5*(yold[i+1] + yold[i-1]) + ...
```

In this case the computer always uses "old" values on the right–hand side, never a mixture of old and new.

It turns out that both approaches are fine. However, the first approach, using a mixture of old and new y values, is easier to implement and is usually faster. This approach is called the *Gauss–Seidel method*. The second approach, which uses only old values for the right–hand side evaluation, is *Jacobi's method*.

Exercise 20.4a

Compare the Gauss–Seidel and Jacobi methods for the hanging cord. In each case, consider the midpoint value of y after 5000 sweeps. For which option does the solution relax most quickly?

Exercise 20.4b

Try a third approach. For each sweep, first step through the odd values of i. This will update the odd `y[i]`'s to new values. Then step through the even values of i, using the updated `y[i]`'s on the right–hand side. How does this approach compare to the Gauss–Seidel and Jacobi methods?

20.5 Speeding up your code

Here is some practical advise. With NumPy's *intrinsic indexing*, the code for Jacobi's method

```
yold = np.copy(y)
for i in range(1,N):
     y[i] = 0.5*(yold[i+1] + yold[i-1]) + ...
```

can be replaced by the single statement

```
y[1:N] = 0.5*(y[2:N+1] + y[0:N-1]) + ...
```

The syntax `y[1:N]` (which can also be written as `y[1:-1]`) denotes the range of elements beginning with `y[1]` and ending with `y[N-1]`. The syntax `y[2:N+1]` (which can also be written as `y[2:]`) denotes the range of elements beginning with `y[2]` and ending with `y[N]`. The syntax `y[0:N-1]` (which can also be written as `y[:-2]`) denotes the range of elements beginning with `y[0]` and ending with `y[N-2]`. Your Jacobi's code will run faster if you use the single statement rather than an explicit `for` loop.

Exercise 20.5a

Solve the hanging cord problem using Jacobi's method with intrinsic indexing. Use the `time()` function from the `time` library to measure the execution time for your code. How much faster does your code run with intrinsic indexing? How does Jacobi's method with intrinsic indexing and 10000 sweeps compare to the Gauss–Seidel method with 5000 sweeps?

Another strategy for speeding up the solution process is called *successive overrelaxation* (SOR). It is a variant of the Gauss–Seidel method. Replace

the finite difference formula (20.15) with the equivalent expression

$$y_i = (1-\omega)y_i + \omega\left[\frac{1}{2}(y_{i+1}+y_{i-1}) - (\mu g/\tau_x)\frac{\Delta x^2}{2}\sqrt{1+(y_{i+1}-y_{i-1})^2/(2\Delta x)^2}\right] . \quad (20.16)$$

For each relaxation sweep, evaluate the right–hand side using the Gauss–Seidel method (a mixture of old and new values of y_{i-1}, y_i and y_{i+1}). The result becomes the new value of y_i.

For $\omega = 1$, the SOR expression (20.16) is identical to the original finite difference formula. For ω larger than 1, but not too large, the relaxation process is more efficient; that is, the numerical solution relaxes more quickly. You can see how quickly the solution is relaxing by monitoring the change in the midpoint value `y[51]` from, say, 4000 sweeps to 5000 sweeps.

Unfortunately, if ω is too large, the SOR algorithm will fail. Determining an appropriate value of ω is a matter of trial and error.

Exercise 20.5b

Apply SOR to the hanging cord problem. Experiment with different values of ω. Does SOR improve the efficiency of your Gauss–Seidel code? What happens if ω is too large? What is the (approximate) maximum value of ω?

20.6 Residual

How many relaxation sweeps should you use? In Sec. 20.3 we noted that the sweeps should continue until the numerical solution satisfies the discrete equation (20.14) to some desired degree of accuracy. We can judge the accuracy of the numerical solution by defining the *residual* at each interior node $i = 1, \ldots, N-1$:

$$\mathcal{R}_i = (y_{i+1} - 2y_i + y_{i-1})/\Delta x^2 - (\mu g/\tau_x)\sqrt{1+(y_{i+1}-y_{i-1})^2/(2\Delta x)^2} . \quad (20.17)$$

As the numerical solution y_i relaxes towards an exact solution of Eq. (20.14), the absolute value of the residual decreases.

Let's assume that at each interior node i, we want $|\mathcal{R}_i|$ to be less than some small number ϵ. This will be the case if the maximum value of $|\mathcal{R}_i|$ is less than ϵ.

Exercise 20.6

Create a code to solve the hanging cord problem and compute the residual after every, say, 100 relaxation sweeps. (Use either the Gauss–Siedel or Jacobi method.) Have your code print out the values of $\max|R_i|$ and continue carrying out sweeps until $\max|R_i|$ drops below some prescribed value ϵ. How many sweeps are required with $\epsilon = 0.001?$ $0.00001?$

20.7 Length of the cord

The numerical solution gives us the shape of the cord for some chosen value of the constant $\mu g/\tau_x$. What is the length of the cord? The length of the curve $y(x)$ is

$$L = \int_{x_a}^{x_b} \sqrt{1 + (dy/dx)^2}\, dx \ , \tag{20.18}$$

as described in basic calculus. Using the differential equation (20.11) we can write this as

$$L = \frac{\tau_x}{\mu g} \int_{x_a}^{x_b} \left(\frac{d^2 y}{dx^2} \right) dx \ , \tag{20.19}$$

then integrate to obtain

$$L = \frac{\tau_x}{\mu g} \left(\frac{dy}{dx} \right) \Bigg|_{x_a}^{x_b} \ . \tag{20.20}$$

The derivatives at the endpoints can be approximated using the two point forward and backward difference formulas (16.4) and (16.6). This gives

$$L \approx \frac{\tau_x}{\mu g \Delta x} \left(y_N - y_{N-1} - y_1 + y_0 \right) \ . \tag{20.21}$$

Exercise 20.7a

Extend your previous code to include a calculation of the cord length L. Use these results to plot a graph of L versus the constant $\mu g/\tau_x$.

This code produces the cord shape $y(x)$ and the cord length L for a given value of $\mu g/\tau_x$. This isn't really what we want. We would like to specify the length L, rather than the constant $\mu g/\tau_x$, and have the code determine the appropriate shape. Here's one strategy for computing the

shape of a hanging cord given its length L and the boundary conditions $y(x_a) = y_a$ and $y(x_b) = y_b$.

Start with a code that computes the cord shape and length for a given constant $\mu g/\tau_x$. For notational simplicity, let $k = \mu g/\tau_x$. Also let L denote the length that you specify, and ℓ denote the numerically computed length. Extend your code to compute the difference $L - \ell$ for a given k. Now imagine that this entire code is a function $\mathcal{F}(k)$ whose input is k and whose output is $L - \ell$. We can use the bisection method of Sec. 11.3 to find the value of k that satisfies $\mathcal{F}(k) = 0$.

To apply bisection, start with k_L and k_R such that $\mathcal{F}(k_L)$ and $\mathcal{F}(k_R)$ have opposite signs. Next, compute the midpoint $k_M = (k_L + k_R)/2$. If $\mathcal{F}(k_M)$ has the same sign as $\mathcal{F}(k_L)$, replace k_L with k_M. If $\mathcal{F}(k_M)$ has the same sign as $\mathcal{F}(k_R)$, replace k_R with k_M. Repeat this process until $|k_R - k_L| < 2\epsilon$, where ϵ is some chosen tolerance. You can take $(k_L + k_R)/2$ as the final answer for the value of k.

What value should you use for the tolerance ϵ? Keep in mind that the relaxation algorithm contains truncation errors whose size depends on the number of grid points. Errors also arise because the number of sweeps is finite, and the numerical solution is never fully relaxed. It would be foolish to set the tolerance too low. Even if the result for k is highly accurate, the shape of the cord will contain errors from these other sources.

Exercise 20.7b

Find the shape of a hanging cord of length $L = 3.0\,\mathrm{m}$ with boundary values $y(0) = 0.0\,\mathrm{m}$ and $y(1.0) = 2.0\,\mathrm{m}$. You might want to use a function definition for the part of your code that computes $\mathcal{F}(k)$. Use bisection to determine the constant $k = \mu g/\tau_x$, and graph the solution $y(x)$.

20.8 Euler–Bernoulli beam theory

Euler–Bernoulli beam theory describes the deflection of an elastic beam in response to a load (forces), assuming the deflection is small. Let the center of the unloaded beam extend along the x–axis. When a load is applied in the y direction, the Euler–Bernoulli equation gives the deflection y away from the x–axis as

$$EI\frac{d^4y}{dx^4} = q(x)\ . \tag{20.22}$$

Here, $q(x)$ is the force per unit length, which can vary along the length of the beam. Hence, q is a function of x. The constant E is the elastic modulus (Young's modulus) of the material. The constant I is the second moment of the cross–section of the beam, defined by

$$I = \iint z^2 \, dy \, dz \ . \tag{20.23}$$

For a square cross section with sides of length ℓ, we have $I = \ell^4/12$.

The Euler–Bernoulli equation is a fourth–order differential equation. For a given beam, the solution depends on the load $q(x)$ as well as the boundary conditions applied at the ends. There are three basic types of boundary conditions: a simple support

$$y\big|_{end} = \text{const} \ , \quad y''\big|_{end} = 0 \ ; \tag{20.24}$$

a clamped end

$$y\big|_{end} = \text{const} \ , \quad y'\big|_{end} = \text{const} \ ; \tag{20.25}$$

and a free end

$$y''\big|_{end} = 0 \ , \quad y'''\big|_{end} = 0 \ . \tag{20.26}$$

Each of the constants should be small, since Euler–Bernoulli theory assumes the deflection is small.

The Euler–Bernoulli equation can be solved numerically using relaxation. Begin by replacing the fourth derivative of y with the centered finite difference approximation

$$y''''(x_i) \approx \frac{y_{i-2} - 4y_{i-1} + 6y_i - 4y_{i+1} + y_{i+2}}{\Delta x^4} \ . \tag{20.27}$$

Then the Euler–Bernoulli equation (20.22) becomes

$$y_i = \frac{1}{6}(-y_{i+2} + 4y_{i+1} + 4y_{i-1} - y_{i-2}) + \frac{q(x_i)}{EI}\frac{\Delta x^4}{6} \ . \tag{20.28}$$

Begin the solution process with a trial solution $y_i = 0$ for $i = 0, \ldots, N$. For each relaxation sweep, evauate the right–hand side of Eq. (20.28) using the current values of y_i. This yields new values for the amplitudes at the interior nodes, $y_2, \ldots, y_{N-2}$. New values for y_0, y_1, y_{N-1} and y_N are obtained from boundary conditions.

To be concrete, let's consider a beam that is cantelievered from the left end. That is, the left end is clamped and the right end is free. The boundary conditions (20.25) at the left end are discretized as

$$y_0 = 0 \ , \quad \frac{y_1 - y_0}{\Delta x} = 0 \ , \tag{20.29}$$

where the two–point forward difference approximation (16.4) is used in place of the first derivative y'. We can solve these equations for y_0 and y_1:

$$y_0 = 0 \ , \tag{20.30a}$$

$$y_1 = 0 \ . \tag{20.30b}$$

The values for y_0 and y_1 remain unchanged from one relaxation sweep to the next.

The right end of the beam is free, and satisfies the boundary conditions (20.26). These conditions can be discretized as

$$\frac{y_N - 2y_{N-1} + y_{N-2}}{\Delta x^2} = 0 \ , \tag{20.31a}$$

$$\frac{y_N - 3y_{N-1} + 3y_{N-2} - y_{N-3}}{\Delta x^3} = 0 \ , \tag{20.31b}$$

and solved for y_{N-1} and y_N:

$$y_N = 3y_{N-2} - 2y_{N-3} \ , \tag{20.32a}$$

$$y_{N-1} = 2y_{N-2} - y_{N-3} \ . \tag{20.32b}$$

As a part of each relaxation sweep, after new values for the interior amplitudes $y_2, \ldots, y_{N-2}$ have been computed, Eqs. (20.32) are used to determine new values for y_N and y_{N-1}.

Exercise 20.8a

Use the methods of Ch. 16 to verify the finite difference formulas (20.27) and (20.31).

Exercise 20.8b

A steel rod has density $\rho = 7850\,\mathrm{kg/m^3}$ and elastic modulus $E = 2.05 \times 10^{11}\,\mathrm{kg/(m \cdot s^2)}$. The rod is 2.0 m long and has a square cross section, $\ell = 0.01\,\mathrm{m}$ on each side. The load is the rod's weight, so the force per unit length is $q(x) = -\rho\ell^2$. The left end of the rod is clamped, the right end is free. Use relaxation to solve the Euler–Bernoulli equation. Monitor the residual. Plot $y(x)$ and determine the deflection at the right end. Hint: SOR is helpful.

Exercise 20.8c

The rod from the previous exercise is now held up with simple supports at each end. Discretize the boundary conditions (20.24) and apply them to your code. Use relaxation to solve the Euler–Bernoulli equation and monitor the residual. Plot $y(x)$. How much does the rod sag in the middle?