Problem 9.7 Evanescent waves

Increase ω past the cutoff frequency in the traveling wave simulation that was found in Problem 9.6. Do you still observe waves? Is energy being transported along the chain? ■

Waves above the cutoff frequency are known as *evanescent waves*. In Problem 9.8 we show how these waves lead to a classical counterpart of quantum mechanical tunneling.

Problem 9.8 Tunneling

(a) Model a traveling wave on an $N = 64$ particle chain with mass $m = 1$ and $k = 1$, but assign $m = 4$ to eight oscillators near the center. Drive the first particle in the chain with a frequency of 0.113. (This value is slightly higher than the frequency above which the wave amplitude falls off exponentially.) Describe the steady-state motion in the left region, the central region of heavier masses, and the right region.

(b) Lower the frequency in part (a) until you observe maximum transmission through the barrier. Describe the steady-state motion in the left region, the central barrier, and the right region. Explain how this system can be used as a frequency filter. ■

9.3 ■ FOURIER SERIES

In Section 9.1 we showed that the displacement of a single particle can be written as a linear combination of normal modes, that is, a linear superposition of sinusoidal terms. In general, an arbitrary periodic function $f(t)$ of period T can be expressed as a sum of sines and cosines:

$$f(t) = \frac{1}{2}a_0 + \sum_{k=1}^{\infty} \big(a_k \cos \omega_k t + b_k \sin \omega_k t\big), \tag{9.24}$$

where

$$\omega_k = k\omega_0 \quad \text{and} \quad \omega_0 = \frac{2\pi}{T}. \tag{9.25}$$

The quantity ω_0 is the fundamental frequency. Such a sum is called a Fourier series. The sine and cosine terms in (9.24) for $k = 2, 3, \ldots$ represent the second, third, ..., and higher-order harmonics. The Fourier coefficients a_k and b_k are given by

$$a_k = \frac{2}{T}\int_0^T f(t) \cos \omega_k t \, dt \tag{9.26a}$$

$$b_k = \frac{2}{T}\int_0^T f(t) \sin \omega_k t \, dt. \tag{9.26b}$$

The constant term $\frac{1}{2}a_0$ in (9.24) is the average value of $f(t)$. The expressions in (9.26) for the coefficients follow from the orthogonality conditions:

$$\frac{2}{T}\int_0^T \sin \omega_k t \sin \omega_{k'} t \, dt = \delta_{k,k'} \tag{9.27a}$$

$$\frac{2}{T}\int_0^T \cos \omega_k t \cos \omega_{k'} t \, dt = \delta_{k,k'} \tag{9.27b}$$

$$\frac{2}{T}\int_0^T \sin \omega_k t \cos \omega_{k'} t \, dt = 0. \tag{9.27c}$$

In general, an infinite number of terms is needed to represent an arbitrary periodic function exactly. In practice, a good approximation usually can be obtained by including a relatively small number of terms. Unlike a power series, which can approximate a function only near a particular point, a Fourier series can approximate a function at almost every point. The `Synthesize` class in Listing 9.3 evaluates such a series given the Fourier coefficients a and b.

Listing 9.3 A class that synthesizes a function using a Fourier series.

```
package org.opensourcephysics.sip.ch09;
import org.opensourcephysics.numerics.Function;

public class Synthesize implements Function {
   // cosine and sine coefficients
   double[] cosCoefficients, sinCoefficients;
   double a0; // the constant term
   double omega0;

   public Synthesize(double period, double a0, double[] cosCoef,
                     double[] sinCoef) {
      omega0 = Math.PI*2/period;
      cosCoefficients = cosCoef;
      sinCoefficients = sinCoef;
      this.a0 = a0;
   }

   public double evaluate(double x) {
      double f = a0/2;
      // sum the cosine terms
      for(int i = 0, n = cosCoefficients.length;i<n;i++) {
         f += cosCoefficients[i]*Math.cos(omega0*x*(i+1));
      }
      // sum the sine terms
      for(int i = 0, n = sinCoefficients.length;i<n;i++) {
         f += sinCoefficients[i]*Math.sin(omega0*x*(i+1));
      }
      return f;
   }
}
```

The `SynthesizeApp` class creates a `Synthesize` object by defining the values of the nonzero Fourier coefficients and draws the result of the Fourier series. Because the `Synthesize` class implements the `Function` interface, we can plot the Fourier series and see how the sum can represent an arbitrary periodic function. An easy way to do so is to create a `FunctionDrawer` and add it to a drawing frame as shown in Listing 9.4. The `FunctionDrawer` handles the routine task of generating a curve from the given function to produce a drawing.