

## Chapter 6

# Oscillations and the Simulation Interface

©2001 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian  
21 February 2002

We introduce threads in the context of exploring the behavior of oscillatory systems and introduce the Simulation interface.

### 6.1 Simple Harmonic Motion

There are many physical systems that undergo regular, repeating motion. Motion that repeats itself at definite intervals, for example, the motion of the earth about the sun, is said to be *periodic*. If an object undergoes periodic motion between two limits over the same path, we call the motion *oscillatory*. Examples of oscillatory motion that are familiar to us from our everyday experience include a plucked guitar string and the pendulum in a grandfather clock. Less obvious examples are microscopic phenomena such as the oscillations of the atoms in crystalline solids.

To illustrate the important concepts associated with oscillatory phenomena, consider an object of mass  $m$  connected to the free end of a spring. The object slides on a frictionless, horizontal surface (see Figure 6.1). We specify the position of the object by  $x$  and take  $x = 0$  to be the *equilibrium* position of the object, that is, the position when the spring is relaxed. If the object is moved from  $x = 0$  and then released, the object oscillates along a horizontal line. If the spring is not compressed or stretched too far from  $x = 0$ , the force on the object at position  $x$  is linearly related to  $x$ :

$$F = -kx. \tag{6.1}$$

The *force constant*  $k$  is a measure of the stiffness of the spring. The negative sign in (6.1) implies that the force acts to restore the object to its equilibrium position. Newton's equation of motion

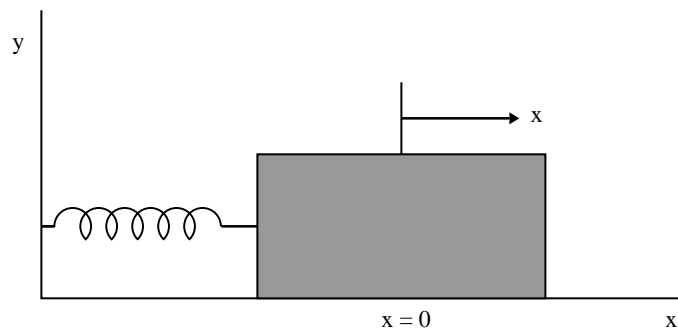


Figure 6.1: A one-dimensional harmonic oscillator. The block slides horizontally on the frictionless surface.

of the object can be written as

$$\frac{d^2x}{dt^2} = -\omega_0^2 x, \quad (6.2)$$

where the angular frequency  $\omega_0$  is defined by

$$\omega_0^2 = \frac{k}{m}. \quad (6.3)$$

The dynamical behavior described by (6.2) is called *simple harmonic motion* and can be solved analytically in terms of sine and cosine functions. Because the form of the solution will help us introduce some of the terminology needed to discuss oscillatory motion, we include the solution here. One form of the solution is

$$x(t) = A \cos(\omega_0 t + \delta), \quad (6.4)$$

where  $A$  and  $\delta$  are constants and the argument of the cosine is in radians. It is straightforward to check by substitution that (6.4) is a solution of (6.2). The constants  $A$  and  $\delta$  are called the *amplitude* and the *phase* respectively, and can be determined by the initial conditions for  $x$  and the velocity  $v = dx/dt$ .

Because the cosine is a periodic function with period  $2\pi$ , we know that  $x(t)$  in (6.4) also is periodic. We define the *period*  $T$  as the smallest time for which the motion repeats itself, that is,

$$x(t + T) = x(t). \quad (6.5)$$

Because  $\omega_0 T$  corresponds to one *cycle*, we have

$$T = \frac{2\pi}{\omega_0} = \frac{2\pi}{\sqrt{k/m}}. \quad (6.6)$$

The *frequency*  $\nu$  of the motion is the number of cycles per second and is given by  $\nu = 1/T$ . Note that  $T$  depends on the ratio  $k/m$  and not on  $A$  and  $\delta$ . Hence the period of simple harmonic motion is independent of the amplitude of the motion.

Although the position and velocity of the oscillator are continuously changing, the total energy  $E$  remains constant and is given by

$$E = \frac{1}{2}mv^2 + \frac{1}{2}kx^2 = \frac{1}{2}kA^2. \quad (6.7)$$

The two terms in (6.7) are the kinetic and potential energies, respectively.

## 6.2 A Simple Program

We will find it convenient to write Newton's equation of motion as a rate equation with the state variables being the position, velocity, and time. Then Newton's equation of motion can be rewritten as a system of *three* coupled first-order equations:

$$\dot{x} = v \quad (6.8a)$$

$$\dot{v} = -\omega_0^2 x \quad (6.8b)$$

$$\dot{t} = 1. \quad (6.8c)$$

Note that the time is being treated in the same way as the other variables. For reasons that will become clearer in Chapter 7, we will represent these elements in the array `state`.

We will follow in the same spirit as the programs in Chapter 5 and first introduce the class `SHO` to represent the mass on the spring:

Listing 6.1: Implementation of the Euler algorithm for the simple harmonic oscillator.

```
package org.opensourcephysics.sip.ch6;

public class SHO {
    double omega2 = 1.0;           // omega2 = k/m
    double[] state = new double[3]; // current state, [ position, velocity, time]

    public void move(double dt) {
        double x = state[0];
        state[0] = state[0] + state[1] * dt; // position
        state[1] = state[1] - omega2 * x * dt; // velocity
        state[2] = state[2] + dt;           // time
    }
}
```

The main difference is that we have represented the position, velocity, and time by a one-dimensional array.

The target application, `SHOApp`, implements the `Calculation` interface in the usual way. Note the similarities with Listing 5.4.

Listing 6.2: Implementation of the `Calculation` interface for the simple harmonic oscillator.

```
package org.opensourcephysics.sip.ch6;
import java.awt.Color;
```

```

import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;

public class SHOApp implements Calculation {
    Control myControl;
    DrawingFrame plottingFrame;
    PlottingPanel plottingPanel;
    DatasetCollection odeData;
    SHO sho;
    double dt;
    int numberOfSteps;
    int datasetIndex = -1;

    public SHOApp() {
        plottingPanel = new PlottingPanel("Time", "position", "Position vs. Time");
        plottingFrame = new DrawingFrame(plottingPanel);
        odeData = new DatasetCollection();
        plottingPanel.addDrawable(odeData);
        sho = new SHO();
    }

    public void setControl(Control control) {
        myControl = control;
        resetCalculation ();
    }

    public void calculate() {
        sho.state [0] = myControl.getDouble("x");
        sho.state [1] = myControl.getDouble("vx");
        numberOfSteps = myControl.getInt("number of steps");
        dt = myControl.getDouble("dt");
        datasetIndex++;
        for(int i = 0; i < numberOfSteps; i++) {
            odeData.append(datasetIndex, sho.state[2], sho.state [0]);
            sho.move(dt);
        }
        plottingPanel.repaint ();
    }

    public void resetCalculation() {
        odeData.clear();
        plottingPanel.repaint ();
        datasetIndex = -1;
        sho.state [0] = 1.0;           // x
        sho.state [1] = 0;           // vx
        sho.state [2] = 0;           // time
        dt = 0.05;
        myControl.setValue("x", sho.state [0]);
        myControl.setValue("vx", sho.state [1]);
        myControl.setValue("dt", dt);
    }
}

```

```

    myControl.setValue("number of steps", 100);
}

public static void main(String[] args) {
    Calculation app = new SHOApp();
    Control c = new CalculationControl(app);
    app.setControl(c);
}
}

```

In Exercise 6.1 we test various algorithms for finding the dynamical motion for the simple harmonic oscillator.

*Problem 6.1.* Verification of SHO

- a. Test `SHOApp` for various values of the time step  $\Delta t$ . Is there anything wrong with the numerical solution?
- b. Define a new class, `SHOexact`, that calculates the analytical solution (6.4) for the simple harmonic oscillator. Instantiate this class in your program and add a plot of the analytical solution to the plotting panel. Describe the qualitative differences between the analytical and numerical solutions. Can the Euler algorithm be rescued by going to smaller step sizes?
- c. Change the `move` method in `SHO` as follows:

```

public void calculate() {
    state[1] = state[1] - omega2*state[0]*dt; // updated velocity
    state[0] = state[0] + state[1]*dt;      // updated position
    state[2] = state[2] + dt;              // time
}

```

Note that `move` no longer implements Euler's algorithm because the velocity is updated first and this new velocity is used to update the position. In contrast, the Euler algorithm requires that all kinematical quantities be updated using the values at the beginning of the interval. As discussed in Chapter 5, the above listing is an implementation of the *Euler-Cromer* algorithm. Does this algorithm have any advantage over the Euler algorithm for this dynamical system?

- d. Modify the program so that  $\omega_0$  is an input variable. Determine the period  $T$  for different values of  $\omega_0$ , and confirm that  $T = 2\pi/\omega$ ?

The results of Problem 6.1 should help us be aware that numerical methods have a *global truncation error*. This global error accumulates not only because each iteration has its own error, but because errors introduced in one iteration can grow in succeeding iteration. In fact, errors in the Euler method often grow exponentially and so merely choosing a smaller time step might not lead to a better solution. In Problem 6.2 we use conservation of energy to determine which algorithm does the best job of solving Newton's equations of motion of the simple harmonic oscillator using a reasonable choice of  $\Delta t$ .

*Problem 6.2.* Energy conservation

- a. Modify your program so that  $E_n$ , the total energy per unit mass, is computed at time  $t_n = t_0 + n\Delta t$ . Use Euler's algorithm and plot the difference  $\Delta E_n = E_n - E_0$  as a function of  $t_n$  for several cycles for a given value of  $\Delta t$ . ( $E_0$  is the initial total energy.) Choose  $x(t=0) = 1$ ,  $v(t=0) = 0$  and  $\omega_0^2 = k/m = 9$  and start with  $\Delta t = 0.05$ . Is the difference  $\Delta E_n$  uniformly small throughout the cycle? Does  $\Delta E_n$  drift, that is, become bigger with time? What is the optimum choice of  $\Delta t$ ?
- b. Use the Euler-Cromer algorithm to answer the same questions as in part (a).
- c. Modify the program so that the Euler-Richardson algorithm is used and answer the same questions as in part (a).
- d. Compute  $\Delta E_n$  and describe the qualitative difference between the time dependence of  $\Delta E_n$  using the three algorithms. Which method is most consistent with the requirement of conservation of energy? For fixed  $\Delta t$ , which algorithm yields better results for the position in comparison to the analytical solution (6.4)? Is the requirement of conservation of energy consistent with the relative accuracy of the computed positions? For which algorithm does the total energy drift the least?
- e. Choose the best algorithm based on the criteria of your choice and determine the values of  $\Delta t$  that are needed to conserve the total energy to within 0.1% over one cycle for  $\omega_0 = 3$  and for  $\omega_0 = 6$ . Can you use the same value of  $\Delta t$  for both values of  $\omega_0$ ? If not, how do the values of  $\Delta t$  correspond to the relative values of the period in the two cases?
- f. It might occur to you that adding a correction for the acceleration to the Euler method would produce more accurate results. The *Verlet* algorithm is based on this idea. One form of this algorithm is given by:

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2, \quad (6.9a)$$

and

$$v_{n+1} = v_n + \frac{1}{2} (a_{n+1} + a_n) \Delta t. \quad (6.9b)$$

Modify the program so that the Verlet algorithm is used.

You might have noticed that modifying your program each time so that a different algorithm can be tested is not very object oriented. In Chapter 7 we will introduce the ODE interface to allow changes in the algorithm to be made easily.

### 6.3 Threads

In Chapter 5 and Section 6.2 we simulated the motion of a particle by solving Newton's equations of motion using several simple numerical methods. The solutions that we obtained were plotted, but we did not attempt to visualize the motion as it occurred, and all our plots were generated *after* the calculations were completed. You probably also noticed that our programs ran for a number of time steps that was determined by a while statement or predetermined by the user.

Hence, we could not intervene to stop the simulation or change a parameter. In the following we will find that in order to visualize the motion as it occurs or to intervene during the simulation to change a parameter, it is necessary to use *threads*.

The `Calculation` interface has an important limitation — it is unable to respond to the click of a button while the calculation is being performed. This lack of response is not a problem if the calculation is short. However, if the calculation takes a long time, the user might assume that the program has crashed or is an infinite loop.

As a simple example of a program that has a similar structure, Listing 6.3 has a Stop button and an associated `ActionListener` whose sole function is to terminate the loop when the button is clicked. The boolean variable, `running`, becomes `true` when the Go button is clicked and should become `false` when the Stop button is clicked. Does this program work as advertised?

Listing 6.3: Example of a program that needs a thread.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

// program will run forever – need a thread to start and stop
public class NeedThreadApp extends JFrame implements ActionListener {
    JButton button;
    boolean running = false;

    public NeedThreadApp() {
        button = new JButton("Go");
        button.addActionListener(this);
        JTextArea outputArea = new JTextArea(20,20);
        Container c = getContentPane();
        c.setLayout(new FlowLayout());
        c.add(button);
        setSize(200,200);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("Go")) {
            button.setText("Stop");
            int i = 0;
            running = true;
            while (running) {
                System.out.println("i = " + i);
                i++;
            }
        }
        if (e.getActionCommand().equals("Stop")) {
            button.setText("Go");
            running = false;
        }
    }
}
```

```
    }  
  
    public static void main(String[] args) {  
        new NeedThreadApp();  
    }  
}
```

### Exercise 6.3. Event queue

Compile the program in Listing 6.3 and run it. What happens if you click on the Go button? How can you stop the program? Your answer will be operating system dependent.

The problem with the program in Listing 6.3 is that once the program enters the while loop, it never leaves the loop to check on the value of `running`. There is no way the program can “listen” for the `stop` button click. What is the value of `running` after the Go button is clicked? In this case the program is trying to do two tasks at once, and there is no way for the program to break out of the while loop. It clearly is poor programming practice not to be able to interrupt a program.

The elegant way of overcoming this problem is to use *threads*. A thread is a single sequential flow of control within a program. Every program has at least one (default) thread. This thread begins at the first statement of `main()`. Traditional procedural programs have one thread. That is, the program starts by executing a line of code, then another line, and then another. The program may jump from one block of code to another, but the program always moves from one statement to the next and never enters a state where statements are executed independently. Java does not have to be like that.

If a computer has only one processor, simultaneous execution of multiple applications is an illusion created by the computer’s operating system. Although the operating system of single processor computer may be able to run multiple applications, it does so by allocating a fraction of a second to one application and then to the another so that it appears that these applications are running simultaneously. In this way, we can scroll pages in our editor and print at apparently the same time. One of the advantages of Java is that it makes simultaneous execution relatively easy.

Multiple threads are used to isolate tasks and allow the program to execute these tasks at the same time. Having multiple threads is similar to having multiple applications insofar as the operating system causes one thread to run and then another. But threads are different from multiple applications; threads are *independent* calculations (not simultaneous calculations) that access the same data. Java may run one thread to produce a graph and another thread to calculate the solution of a differential equation. In fact, the graph may be plotting the numerical solution to the equation. Because threads can share the same data, we have to make sure that these calculations work together in such a way that neither calculation changes the data that is being used by the other. One of the advantages of Java is that multiple threads are automatically distributed over multiple processors.

A thread can execute statements within an object that implements the `Runnable` interface. This interface consists of a single method, the `run` method, and the thread executes the code within this method. The `run` method cannot be invoked directly. When the `run` method returns, the calculation thread stops executing and is said to die. Another thread must be created if we wish to invoke the `run` method a second time. Listing 6.4 shows how threads can be used to start and stop a calculation.

Listing 6.4: Simple example of the use of threads.

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

// uses thread to start and stop
public class FirstThreadApp extends JFrame implements ActionListener, Runnable
{
    JButton button;
    boolean running = false;
    Thread thread;

    public FirstThreadApp() {
        button = new JButton("Go");
        button.addActionListener(this);
        Container c = getContentPane();
        c.setLayout(new FlowLayout());
        c.add(button);
        setSize(200,200);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("Go")) {
            button.setText("Stop");
            running = true;
            thread = new Thread(this);
            thread.start();
        }
        if (e.getActionCommand().equals("Stop")) {
            button.setText("Go");
            running = false;
        }
    }

    public void run() {
        int i = 0;
        while (running) {
            System.out.println("i = " + i);
            i++;
        }
    }

    public static void main(String[] args) {
        new FirstThreadApp();
    }
}

```

Note that the program is a `Runnable` object. The thread is created inside the `ActionPerformed`

method by invoking the thread's constructor, `thread = new Thread(this)`, and passing a reference to the `Runnable` program. The thread will start running after the `main` method is executed.

Writing a proper `run` method is not difficult, but requires attention to detail. The most important consideration is how to end the loop, thereby stopping the calculation and killing the thread. The correct way to stop the `run` method is to set the calculation thread to null. It is important that a thread not monopolize the computer's processor(s) without giving other threads a chance to run. We can do so by invoking the `sleep(int delay)` method from within the `run` method. The argument that is passed to `sleep` is the number of milliseconds that the thread should wait before continuing the calculation. A good value for computational speed is 10 milliseconds. Because the eye cannot respond to rapid screen redraws, a good value for smooth animation is 100 milliseconds or 1/10 of a second. Note that Java requires that the `sleep` method be enclosed in a `try` block to properly handle interrupt exceptions.

If possible, we should avoid having two threads accessing the same data. If a thread changes an object's data, we must insure that this change does not effect another concurrent use of the same data. Our programs take the easy way out by creating a new thread only if the previous calculation thread has died.

## 6.4 Animation Interface

In order to start and stop threads and hence be able to do animations, we have designed a more flexible graphical user interface, the `Animation` interface, to complement the `Calculation` interface that we introduced in Chapter 4. The `AnimationControl` class implements the `Control` interface and is similar to the `Control` interface that we have been using. The `AnimationControl` class has six buttons that invoke five methods. These methods are defined in another class that implements the `Animation` interface.

Listing 6.5: The `Animation` interface.

```
package org.opensourcephysics.controls;

public interface Animation extends Runnable {

    public void setControl(Control control);
    public void startAnimation();
    public void stopAnimation();
    public void initializeAnimation();
    public void resetAnimation();
    public void stepAnimation();
}
```

The methods in the `Animation` interface initialize, start, stop, step, and reset a calculation.

### *Exercise 6.4.* Animation control test

The Chapter 6 directory has a test program, `AnimationTestApp`. Copy this program into a working directory, edit the package declaration and the import statements, and then compile and run this application. Describe the sequence of actions that the user must perform to run the calculation. What happens to the parameter input area in the control when the calculation is running?

The most important function of the `AnimationControl` class is to handle input/output by invoking the `initializeAnimation` and `resetAnimation` methods. Note that starting an animation is a two-step process. When an animation is created, it is in input mode. Input mode is used to edit parameter values, whereas run mode is used to run and single step a calculation. Consequently, the parameters to be changed in run mode. It is possible to return to input mode at any time by clicking the `edit` button. This paradigm may seem constraining at first, but it insures that input parameters are clearly stated and that these parameter are not changed arbitrarily while the program is running.

To gain familiarity with the animation user interface, we recast the simple harmonic oscillator program into this framework

Listing 6.6: Simple harmonic oscillator using the Animation interface.

```

package org.opensourcephysics.sip.ch6;
import java.awt.Color;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;

public class SHOThreadApp implements Animation {
    Control myControl;
    DrawingFrame plottingFrame;
    PlottingPanel plottingPanel;
    DatasetCollection odeData;
    SHO sho;
    double dt;
    Thread animationThread;

    public SHOThreadApp() {
        plottingPanel = new PlottingPanel("time", "position", "position vs. time");
        plottingFrame = new DrawingFrame(plottingPanel);
        odeData = new DatasetCollection();
        plottingPanel.addDrawable(odeData);
        sho = new SHO();
    }

    public void setControl(Control control) {
        myControl = control;
        resetAnimation();
    }

    public void initializeAnimation() {
        sho.state[0] = myControl.getDouble("x");
        sho.state[1] = myControl.getDouble("vx");
        dt = myControl.getDouble("dt");
    }

    public void startAnimation() {
        animationThread = new Thread(this);
        animationThread.start();
    }
}

```

```

public void stopAnimation() {
    animationThread = null;
    updateControlValues();
}

public void run() {
    while (animationThread == Thread.currentThread()) {
        odeData.append(0, sho.state[2], sho.state [0]);           // first dataset
        sho.move(dt);
        plottingPanel.paintImmediately(plottingPanel.getBounds()); // paint entire panel
        try {
            Thread.sleep(10);
        } catch (InterruptedException e) {}
    }
}

public void stepAnimation() {
    odeData.append(0, sho.state[2], sho.state [0]);
    sho.move(dt);
    plottingPanel.repaint ();
    updateControlValues();
}

public void resetAnimation() {
    odeData.clear();
    plottingPanel.repaint ();
    sho.state [0] = 1.0;
    sho.state [1] = 0;
    sho.state [2] = 0;
    dt = 0.2;
    myControl.setValue("dt", dt);
    updateControlValues();
}

public void updateControlValues() {
    myControl.setValue("x", sho.state [0]);
    myControl.setValue("vx", sho.state [1]);
}

public static void main(String[] args) {
    Animation app = new SHOThreadApp();
    Control myControl = new AnimationControl(app);
    app.setControl(myControl);
}
}

```

Running the program and clicking the `init` button changes the behavior of the control. The first button now changes its label from `Init` to `start`. Clicking the `start` button creates the animation thread. The heart of the animation is the `run` method because it performs the physics

and transfers the resulting data to the display objects. The `sho` object encapsulates the oscillator's state variables and a solution method. The `sho.step(dt)` statement advances the state by  $\Delta t$ .

You may want to examine the current values as you run an animation. One way to do this is to update the control when the calculation is stopped. These values can then be edited. Another way is to write current values using the `System.out.println` or `myControl.println` methods. The `resetAnimation` method should either stop a calculation if it is currently in run mode or reset the simulation to a known initial state if it is not. The `stepAnimation` method is designed to allow us to single step an animation. Because we often want to examine how the state changes in a single step, we write the current values or position, velocity, and time to the control as we did in the `stopAnimation` method. We now have all the pieces needed to construct fairly sophisticated simulation program.

*Problem 6.5.* Analysis of simple harmonic motion

1. Modify the simple harmonic oscillator program so that the position and velocity of the oscillator are plotted as a function of the time  $t$ . Choose one of the four numerical algorithms and the optimum value of  $\Delta t$  that you determined in Problem 6.2. Describe the qualitative behavior of the position and velocity.
2. Plot the time dependence of the potential energy and the kinetic energy through one complete cycle. Where in the cycle is the kinetic energy a maximum?
3. Compute the average value of the kinetic energy and the potential energy during a complete cycle. Is there a relation between the two averages?
4. Compute  $x(t)$  for different values of  $A$  and show that the shape of  $x(t)$  is independent of  $A$ , that is, show that  $x(t)/A$  is a *universal* function of  $t$  for a fixed value of  $k/m$ . In what units should the time be measured so that the ratio  $x(t)/A$  is independent of  $A$  and  $k/m$ ?
5. The state of motion of the one-dimensional oscillator is completely specified as a function of time by  $x(t)$  and  $p(t)$ , where  $p$  is the momentum of the oscillator. These quantities may be interpreted as the coordinates of a point in a two-dimensional space known as *phase space*. As time increases, the point  $(x(t), p(t))$  moves along a *trajectory in phase space*. Modify your program so that the momentum  $p$  is plotted as a function of  $x$ , that is, choose  $p$  and  $x$  as the vertical and horizontal axes respectively. Set  $\omega_0 = 3$  and compute the phase space trajectories for the initial condition  $x(t = 0) = 1, v(t = 0) = 0$ . What is the shape of the trajectory in phase space? What is the shape for the initial conditions,  $x(t = 0) = 0, v(t = 0) = 1$  and  $x(t = 0) = 4, v(t = 0) = 0$ ? Do you find a different phase trajectory for each initial condition? What physical quantity distinguishes the phase trajectories? Is the motion of a representative point  $(x, p)$  always in the clockwise or counterclockwise direction?

Threads can be used in any situation where the number of repetitions is not predetermined and the user is expected to interact with a visualization, such as in Problems 6.6 and 6.7.

*Problem 6.6.* Lissajous figures

A computer display can be used to simulate the output seen on an oscilloscope. Imagine that the vertical and horizontal inputs to an oscilloscope are sinusoidal in time, that is,  $x = A \sin(\omega_x t + \phi_x)$  and  $y = B \sin(\omega_y t + \phi_y)$ . If the curve that is drawn repeats itself, such a curve is called a *Lissajous*

*figure.* For what values of the angular frequencies  $\omega_x$  and  $\omega_y$  do you obtain a Lissajous figure? How do the phase factors  $\phi_x$  and  $\phi_y$  and the amplitudes  $A$  and  $B$  affect the curves? First choose  $A = B = 1$ ,  $\omega_x = 2$ ,  $\omega_y = 3$ ,  $\phi_x = \pi/6$ , and  $\phi_y = \pi/4$ . Write a program to plot  $y$  versus  $x$ , as  $t$  advances from  $t = 0$ .

Traveling waves are ubiquitous in nature and give rise to a important phenomena such as beats and standing waves. We investigate their behavior in Problems 6.7.

*Problem 6.7. Superposition of waves*

- Write a program to plot  $A \sin(kx + \omega t)$  from  $x = x_{\min}$  to  $x = x_{\max}$  for various values of  $t$  as  $t$  advances from  $t = 0$  to  $2\pi/\omega$ . For simplicity, take  $A = 1$ ,  $\omega = 2\pi$ , and  $\lambda = 2\pi/k = 2$ .
- Use your program to demonstrate a standing wave with wavelength  $\lambda = 2$  and frequency of unity. What function should you plot?
- Use your program to demonstrate beats by plotting  $(y_1 + y_2)^2$  as functions of time in the range  $x_{\min} = -10$  and  $x_{\max} = 10$ . The quantity  $(y_1 + y_2)^2$  corresponds to the superpositions of two waves. What is the beat frequency for each of the superpositions?

$$y_1(x, t) = \sin[8.4(x - 1.1t)] \quad (6.10a)$$

$$y_2(x, t) = \sin[8.0(x - 1.1t)] \quad (6.10b)$$

and

$$y_1(x, t) = \sin[8.4(x - 1.2t)] \quad (6.11a)$$

$$y_2(x, t) = \sin[8.0(x - 1.0t)] \quad (6.11b)$$

and

$$y_1(x, t) = \sin[8.4(x - 1.0t)] \quad (6.12a)$$

$$y_2(x, t) = \sin[8.0(x - 1.2t)]. \quad (6.12b)$$

What difference do you observe between these superpositions?

## 6.5 Visualization of the Motion of a Pendulum

A common example of a mechanical system that exhibits oscillatory motion is the simple pendulum (see Figure 6.2). A simple pendulum is an idealized system consisting of a particle or bob of mass  $m$  attached to the lower end of a rigid rod of length  $L$  and negligible mass; the upper end of the rod pivots without friction. If the bob is pulled to one side from its equilibrium position and released, the pendulum swings in a vertical plane.

Because the bob is constrained to move along the arc of a circle of radius  $L$  about the center  $O$ , the bob's position is specified by the arc length or by the angle  $\theta$  (see Figure 6.2). The linear velocity and acceleration of the bob as measured along the arc are given by

$$v = L \frac{d\theta}{dt} \quad (6.13)$$

$$a = L \frac{d^2\theta}{dt^2}. \quad (6.14)$$

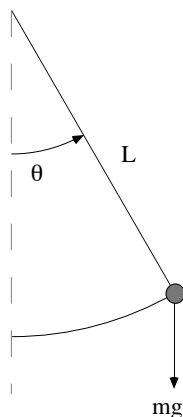


Figure 6.2: Force diagram for a simple pendulum. The angle  $\theta$  is measured from the vertical direction and is positive if the mass is to the right of the vertical and negative if it is to the left.

In the absence of friction, two forces act on the object: the force  $mg$  vertically downward and the force of the rod which is directed inward to the center if  $|\theta| < \pi/2$ . Note that the effect of the rigid rod is to constrain the motion of the bob along the arc. From Figure 6.2, we can see that the component of  $mg$  along the arc is  $mg \sin \theta$  in the direction of decreasing  $\theta$ . Hence, the equation of motion can be written as

$$mL \frac{d^2\theta}{dt^2} = -mg \sin \theta \quad (6.15)$$

or

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L} \sin \theta. \quad (6.16)$$

Equation (6.16) is an example of a nonlinear equation because  $\sin \theta$  rather than  $\theta$  appears. Most nonlinear equations do not have analytical solutions in terms of well-known functions, and (6.16) is no exception. However, if the pendulum undergoes oscillations of sufficiently small amplitude, then  $\sin \theta \approx \theta$ , and (6.16) reduces to

$$\frac{d^2\theta}{dt^2} \approx -\frac{g}{L} \theta. \quad (\theta \ll 1) \quad (6.17)$$

Remember that  $\theta$  is measured in radians.

Part of the fun of studying physics comes from realizing that equations that appear in different areas (and different fields) are often identical. An example of this “crossover” effect can be seen from a comparison of (6.2) and (6.17). If we associate  $x$  with  $\theta$ , we see that the two equations are identical in form, and we can immediately conclude that for  $\theta \ll 1$ , the period of a pendulum is given by

$$T = 2\pi\sqrt{L/g}. \quad (6.18)$$

One way to understand the motion of a pendulum with large oscillations is to solve (6.16) numerically. Because we know that a numerical solution must be consistent with conservation of total energy, we derive its form here. The potential energy can be found from the following considerations. If the rod is deflected by the angle  $\theta$ , then the bob is raised by the distance  $h = L - L \cos \theta$  (see Figure 6.2). Hence, the potential energy of the bob in the gravitational field of the earth can be expressed as

$$U = mgh = mgL(1 - \cos \theta), \quad (6.19)$$

where the zero of the potential energy corresponds to  $\theta = 0$ . Because the kinetic energy of the pendulum is  $\frac{1}{2}mv^2 = \frac{1}{2}mL^2(d\theta/dt)^2$ , the total energy  $E$  is

$$E = \frac{1}{2}mL^2\left(\frac{d\theta}{dt}\right)^2 + mgL(1 - \cos \theta). \quad (6.20)$$

In the following, we adopt the notation  $\omega = d\theta/dt$  for the angular velocity.

We use two classes to solve simulate and visualize the motion of a pendulum problem, `Pendulum` and `PendulumApp`. The `Pendulum` class, a drawable class that solves the equation of motion, is coded as follows:

Listing 6.7: A Drawable pendulum.

```
package org.opensourcephysics.sip.ch6;
import java.awt.*;
import org.opensourcephysics.display.*;

public class Pendulum implements Drawable {
    protected double omega2; // omega^2 = g/length
    protected double[] state = new double[3]; // current state
    Color color = Color.red;
    int pixRadius = 6;

    public Pendulum() { // create Pendulum with angular frequency 1
        this(1);
    }

    public Pendulum(double _omega2) { // create Pendulum with specified angular frequency
        omega2 = _omega2;
        state[0] = 0; // theta
        state[1] = 0; // angular velocity
        state[2] = 0; // time
    }

    public void setState(double theta, double omega, double t) {
        state[0] = theta;
        state[1] = omega;
        state[2] = t;
    }

    public double[] getState() {
```

```

    return state;
}

public void step(double dt) {
    // use Verlet algorithm
    double accel = -omega2*Math.sin(state[0]);
    state[0] += state[1]*dt + 0.5*accel*dt*dt;           // angle
    state[1] += 0.5*(accel - omega2*Math.sin(state[0]))*dt; // angular velocity
    state[2] += dt;                                     // time
}

public void draw(DrawingPanel drawingPanel, Graphics g) {
    int xpivot = drawingPanel.xToPix(0);
    int ypivot = drawingPanel.yToPix(0);
    int xpix = drawingPanel.xToPix(Math.sin(state[0]));
    int ypix = drawingPanel.yToPix(-Math.cos(state[0]));
    g.setColor(Color.black);
    g.drawLine(xpivot, ypivot, xpix, ypix);
    g.setColor(color);
    g.fillOval(xpix - pixRadius, ypix - pixRadius, 2*pixRadius, 2*pixRadius); // draw bob
}
}

```

Note that the step method implements the Verlet algorithm.

Encapsulating drawing within the `Pendulum` class simplifies the `PendulumApp` code because it is unnecessary for the run method to pass data to a separate drawing object. Stepping the ODE automatically updates the drawing's position.

Listing 6.8: Visualization of the motion of a pendulum.

```

package org.opensourcephysics.sip.ch6;
import java.awt.Color;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;

public class PendulumApp implements Animation {

    Control myControl;
    // contains the graph
    PlottingPanel plottingPanel = new PlottingPanel("Time", "Theta", "angle vs time");
    DrawingFrame plottingFrame = new DrawingFrame(plottingPanel);
    // contains the animation
    DrawingPanel drawingPanel = new DrawingPanel();
    DrawingFrame drawingFrame = new DrawingFrame(drawingPanel);
    Thread animationThread;
    DatasetCollection dataset;
    Pendulum pendulum;
    double dt;

    public PendulumApp() {

```

```
    pendulum = new Pendulum();
    drawingPanel.setPreferredMinMax(-1.2, 1.2, -1.2, 1.2);
    drawingPanel.addDrawable(pendulum);
    dataset = new DatasetCollection();
    dataset.setXYColumnNames(0, "time", "theta");
    plottingPanel.addDrawable(dataset);
}

public void setControl(Control control) {
    myControl = control;
    resetAnimation();
}

public void startAnimation() {
    animationThread = new Thread(this);
    animationThread.start(); // start the animation
}

public void initializeAnimation() {
    dt = myControl.getDouble("dt"); // time step
    double theta = myControl.getDouble("theta");
    double omega = myControl.getDouble("omega");
    double[] state = pendulum.getState();
    pendulum.setState(theta, omega, state[2]);
    plottingPanel.repaint();
    drawingPanel.repaint();
}

public void stopAnimation() {
    Thread runningThread = animationThread;
    animationThread = null; // stop the animation
    try { // wait for the thread to die
        if (runningThread != null) {
            runningThread.join();
        }
    } catch (InterruptedException e) {}
    updateControlValues();
}

public void updateControlValues() {
    double[] state = pendulum.getState();
    myControl.setValue("theta", state[0]);
    myControl.setValue("omega", state[1]);
}

public void stepAnimation() {
    stepPendulum();
    plottingPanel.repaint();
    drawingPanel.repaint();
    updateControlValues();
}
```

```

    }

    public void stepPendulum() {
        double[] state = pendulum.getState(); // get the state
        dataset.append(0, state [2], state [0]); // add data to the curve
        pendulum.step(dt); // advance the SHO state by dt
    }

    public void resetAnimation() {
        dataset.clear ();
        plottingPanel.repaint ();
        drawingPanel.repaint ();
        double[] state = pendulum.getState();
        double theta = 1;
        double omega = 1;
        double t = 0;
        pendulum.setState(theta, omega, t);
        updateControlValues();
        myControl.setValue("dt", 0.1);
    }

    public void run() {
        while(animationThread == Thread.currentThread()) {
            stepPendulum();
            plottingPanel.render (); // paint the plot
            drawingPanel.render (); // paint the animation
            try {
                animationThread.sleep(100);
            } catch(InterruptedException ie) {}
        }
    }

    public static void main(String[] args) {
        Animation app = new PendulumApp();
        AnimationControl control = new AnimationControl(app);
        app.setControl(control);
    }
}

```

*Problem 6.8.* Oscillations of a pendulum

1. Modify the `Pendulum` class so that the acceleration is specified in its own method. That is, modify the rate equations in the `step` method so that they use the following method:

```

private double getAcceleration(){
    return -omega2*Math.sin(state[0]);
}

```

This change will make it easier to add friction and an external driving force in subsequent problems.

2. Make the necessary changes so that the analytical solution for small angles is also plotted.
3. Test the program at sufficiently small amplitude so that  $\sin \theta \approx \theta$ . Choose  $g/L = 9$  and the initial condition  $\theta(t = 0) = 0.1$ ,  $\omega(t = 0) = 0$  and determine the period. Estimate the error due to the small angle approximation for these initial conditions.
4. Simulate large amplitude oscillations of a pendulum. Set  $g/L = 9$  and choose  $\Delta t$  so the numerical algorithm generates a stable solution. Check the stability of the solution by monitoring the total energy and ensuring that it does not drift from its initial value.
5. Set  $\omega(t = 0) = 0$  and make plots of  $\theta(t)$  and  $\omega(t)$  for the initial conditions  $\theta(t = 0) = 0.1, 0.2, 0.4, 0.8,$  and  $1.0$ . Remember that  $\theta$  is measured in radians. Describe the qualitative behavior of  $\theta$  and  $\omega$ . What is the period  $T$  and the amplitude  $\theta_m$  in each case? Plot  $T$  versus  $\theta_m$  and discuss the qualitative dependence of the period on the amplitude. How do your results for  $T$  compare in the linear and nonlinear cases, or example, which period is larger? Explain the relative values of  $T$  in terms of the relative magnitudes of the restoring force in the two cases.

## 6.6 Damped Harmonic Oscillator

We know from experience that most oscillatory motion in nature gradually decreases until the displacement becomes zero; such motion is said to be *damped* and the system is said to be *dissipative* rather than conservative. As an example of a damped harmonic oscillator, consider the motion of the block in Figure 6.1 when a horizontal drag force is included. For small velocities, it is a reasonable approximation to assume that the drag force is proportional to the first power of the velocity. In this case the equation of motion can be written as

$$\frac{d^2x}{dt^2} = -\omega_0^2 x - \gamma \frac{dx}{dt}. \quad (6.21)$$

The *damping coefficient*  $\gamma$  is a measure of the magnitude of the drag term. Note that the drag force in (6.21) opposes the motion. We simulate the behavior of the damped linear oscillator in Problem 6.9.

*Problem 6.9.* Damped linear oscillator

1. Incorporate the effects of damping into your harmonic oscillator animation and plot the time dependence of the position and the velocity. Describe the qualitative behavior of  $x(t)$  and  $v(t)$  for  $\omega_0 = 3$  and  $\gamma = 0.5$  with  $x(t = 0) = 1, v(t = 0) = 0$ .
2. The period of the motion is the time between successive maxima of  $x(t)$ . Compute the period and corresponding angular frequency and compare their values to the undamped case. Is the period longer or shorter? Make additional runs for  $\gamma = 1, 2,$  and  $3$ . Does the period increase or decrease with greater damping? Why?
3. The amplitude is the maximum value of  $x$  during one cycle. Compute the *relaxation time*  $\tau$ , the time it takes for the amplitude of an oscillation to decrease by  $1/e \approx 0.37$  from its maximum value. Is the value of  $\tau$  constant throughout the motion? Compute  $\tau$  for the values of  $\gamma$  considered in part (b) and discuss the qualitative dependence of  $\tau$  on  $\gamma$ .

4. Plot the total energy as a function of time for the values of  $\gamma$  considered in part (b). If the decrease in energy is not monotonic, explain.
5. Compute the time dependence of  $x(t)$  and  $v(t)$  for  $\gamma = 4, 5, 6, 7,$  and  $8$ . Is the motion oscillatory for all  $\gamma$ ? How can you characterize the decay? For fixed  $\omega_0$ , the oscillator is said to be *critically damped* at the smallest value of  $\gamma$  for which the decay to equilibrium is monotonic. For what value of  $\gamma$  does critical damping occur for  $\omega_0 = 4$  and  $\omega_0 = 2$ ? For each value of  $\omega_0$ , compute the value of  $\gamma$  for which the system approaches equilibrium most quickly.
6. Compute the phase space diagram for  $\omega_0 = 3$  and  $\gamma = 0.5, 2, 4, 6,$  and  $8$ . Why does the phase space trajectory converge to the origin,  $(x = 0, v = 0)$ ? This point is called an *attractor*. Convince yourself that points near the origin move toward the attractor; this attractor is said to be *stable*. Are these qualitative features of the phase space plot independent of  $\gamma$ ?

*Problem 6.10. Damped nonlinear pendulum*

Consider a damped pendulum with  $g/L = 9$  and  $\gamma = 1$  and the initial condition  $\theta(t = 0) = 0.2, \omega(t = 0) = 0$ . In what ways is the motion of the damped nonlinear pendulum similar to the damped linear oscillator? In what ways is it different? What is the shape of the phase space trajectory for the initial condition  $\theta(t = 0) = 1, \omega(t = 0) = 0$ ? Do you find a different phase trajectory for other initial conditions? Remember that  $\theta$  is restricted to be between  $-\pi$  and  $+\pi$ .

## 6.7 Response to External Forces

How can we determine the period of a pendulum that is not already in motion? The obvious way is to disturb the system, for example, to displace the bob and observe its motion. We will find that the nature of the *response* of the system to the perturbation tells us something about the nature of the system in the absence of the perturbation.

Consider the *driven* damped linear oscillator with an external force  $F(t)$  in addition to the linear restoring force and linear damping force. The equation of motion can be written as

$$\frac{d^2x}{dt^2} = -\omega_0^2x - \gamma v + \frac{1}{m}F(t). \quad (6.22)$$

It is customary to interpret the response of the system in terms of the displacement  $x$  rather than the velocity  $v$ .

The time dependence of  $F(t)$  in (6.22) is arbitrary. Because many forces are periodic, we first consider the form

$$\frac{1}{m}F(t) = A_0 \cos \omega t, \quad (6.23)$$

where  $\omega$  is the angular frequency of the driving force. In Problem 6.11 we consider the response of the damped linear oscillator to (6.23).

*Problem 6.11. Motion of a driven damped linear oscillator*

1. Modify the harmonic oscillator animation so that an external force of the form (6.23) is included. Add this force in the class that encapsulates the equations of motion without changing the target class. The angular frequency of the driving force,  $\omega$  should be added as an input parameter.
2. Set  $\omega_0 = 3$ ,  $\gamma = 0.5$ ,  $\omega = 2$  and the amplitude of the external force  $A_0 = 1$  for all runs unless otherwise stated. We know that for these values of  $\omega_0$  and  $\gamma$ , the dynamical behavior in the absence of an external force corresponds to a underdamped oscillator. Plot  $x(t)$  versus  $t$  in the presence of the external force with the initial condition,  $x(t = 0) = 1, v(t = 0) = 0$ . How does the qualitative behavior of  $x(t)$  differ from the nonperturbed case? What is the period and angular frequency of  $x(t)$  after several oscillations have occurred? Repeat the same observations for  $x(t)$  with  $x(t = 0) = 0, v(t = 0) = 1$ . Does  $x(t)$  approach a limiting behavior independently of the initial conditions? Does the short time behavior of  $x(t)$  depend on the initial conditions? Identify a *transient* part of  $x(t)$  that depends on the initial conditions and decays in time, and a *steady state* part that dominates at longer times and is independent of the initial conditions.
3. Compute  $x(t)$  for several combinations of  $\omega_0$  and  $\omega$ . What is the period and angular frequency of the steady state motion in each case? What parameters determine the frequency of the steady state behavior?
4. One measure of the long-term behavior of the driven harmonic oscillator is the amplitude of the steady state displacement  $A(\omega)$ . The amplitude function  $A(\omega)$  can be computed easily if we set  $\omega$  and run the system until steady state has been achieved. We then test the position after every time step to see if a new maximum has been reached.

```

if (x > Math.abs(max)) {
    max = Math.abs(x);
    System.out.println("new max =" + max);
}

```

5. Verify that the steady state behavior of  $x(t)$  is given by

$$x(t) = A(\omega) \cos(\omega t + \delta). \quad (6.24)$$

The quantity  $\delta$  is the phase difference between the applied force and the steady state motion. Compute  $A(\omega)$  and  $\delta(\omega)$  for  $\omega_0 = 3$ ,  $\gamma = 0.5$ , and  $\omega = 0, 1.0, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0, 3.2$ , and  $3.4$ . Choose the initial condition,  $x(t = 0) = 0, v(t = 0) = 0$ . Repeat the simulation for  $\gamma = 3.0$ , and plot  $A(\omega)$  and  $\delta(\omega)$  versus  $\omega$  for the two values of  $\gamma$ . Discuss the qualitative behavior of  $A(\omega)$  and  $\delta(\omega)$  for the two values of  $\gamma$ . If  $A(\omega)$  has a maximum, determine the angular frequency  $\omega_m$  at which the maximum of  $A$  occurs. Is the value of  $\omega_m$  close to the natural angular frequency  $\omega_0$ ? Compare  $\omega_m$  to  $\omega_0$  and to the frequency of the damped linear oscillator in the absence of an external force.

6. Compute  $x(t)$  and  $A(\omega)$  for a damped linear oscillator with the amplitude of the external force  $A_0 = 4$ . How do the steady state results for  $x(t)$  and  $A(\omega)$  compare to the case  $A_0 = 1$ ? Does the transient behavior of  $x(t)$  satisfy the same relation as the steady state behavior?

7. What is the shape of the phase space trajectory for the initial condition  $x(t = 0) = 1, v(t = 0) = 0$ ? Do you find a different phase trajectory for other initial conditions?
8. Describe the qualitative behavior of the steady state amplitude  $A(\omega)$  near  $\omega = 0$  and  $\omega \gg \omega_0$ . Why is  $A(\omega = 0) < A(\omega)$  for small  $\omega$ ? Why does  $A(\omega) \rightarrow 0$  for  $\omega \gg \omega_0$ ?
9. Does the mean kinetic energy resonate at the same frequency as does the amplitude? Compute the mean kinetic energy over one cycle once steady state conditions have been reached. Choose  $\omega_0 = 3$  and  $\gamma = 0.5$ .

In Problem 6.11 we found that the response of the damped harmonic oscillator to an external driving force is linear. For example, if the magnitude of the external force is doubled, then the magnitude of the steady state motion also is doubled. This behavior is a consequence of the linear nature of the equation of motion. When a particle is subject to nonlinear forces, the response can be much more complicated (see Section ?? [in Chaos chapter] ).

For many problems, the sinusoidal driving force in (6.23) is not realistic. Another example of an external force can be found by observing someone pushing a child on a swing. Because the force is nonzero only for short intervals of time, this type of force is impulsive. In the following problem, we consider the response of a damped linear oscillator to an impulsive force.

*\*Problem 6.12.* Response of a damped linear oscillator to nonsinusoidal external forces

1. Assume a swing can be modelled by a linear restoring force and a linear damping term. The effect of an impulse is to change the velocity. For simplicity, let the duration of the push equal the time step  $\Delta t$ . Introduce an integer variable for the number of time steps and use the mod function to ensure that the impulse is nonzero only at the time interval associated with the period of the external impulse.
2. Determine the steady state amplitude  $A(\omega)$  for  $\omega = 1.0, 1.3, 1.4, 1.5, 1.6, 2.5, 3.0,$  and  $3.5$ . The corresponding period of the impulse is given by  $T = 2\pi/\omega$ . Choose  $\omega_0 = 3$  and  $\gamma = 0.5$ . Are your results consistent with your experience in pushing a swing and with the comparable results of Problem 6.11?
3. Consider the response to the half-wave external force consisting of the positive part of a cosine function (see Figure 6.3). Compute  $A(\omega)$  for  $\omega_0 = 3$  and  $\gamma = 0.5$ . At what values of  $\omega$  does  $A(\omega)$  have a relative maxima? Is the half-wave cosine driving force equivalent to a sum of cosine functions of different frequencies? For example, does  $A(\omega)$  have more than one resonance?
4. Compute the steady state response  $x(t)$  to the external force

$$\frac{1}{m}F(t) = \frac{1}{\pi} + \frac{1}{2} \cos t + \frac{2}{3\pi} \cos 2t - \frac{2}{15\pi} \cos 4t. \quad (6.25)$$

How does a plot of  $F(t)$  versus  $t$  compare to the half-wave cosine function? Use your results to conjecture a principle of superposition for the solutions to linear equations.

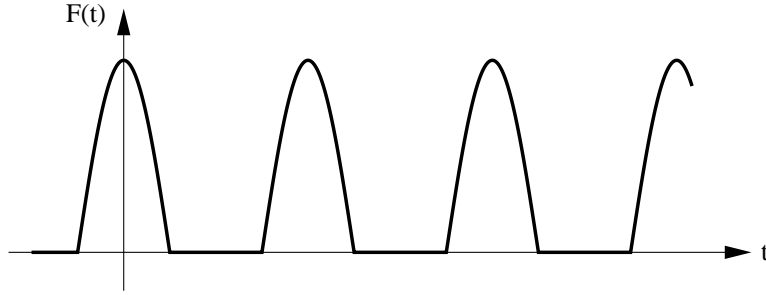


Figure 6.3: A half-wave driving force corresponding to the positive part of a cosine function.

element	voltage drop	units
resistor	$V_R = IR$	resistance $R$ , ohms ( $\Omega$ )
capacitor	$V_C = Q/C$	capacitance $C$ , farads ( $F$ )
inductor	$V_L = L dI/dt$	inductance $L$ , henries ( $H$ )

Table 6.1: The voltage drops across the basic electrical circuit elements.  $Q$  is the charge (coulombs) on one plate of the capacitor, and  $I$  is the current (amperes).

## 6.8 Electrical Circuit Oscillations

In this section we discuss several electrical analogues of the mechanical systems we have considered. Although the equations of motion are identical in form, it is convenient to consider electrical circuits separately, because the nature of the questions is somewhat different.

The starting point for electrical circuit theory is Kirchhoff's loop rule, which states that the sum of the voltage drops around a closed path of an electrical circuit is zero. This law is a consequence of conservation of energy, because a voltage drop represents the amount of energy that is lost or gained when a unit charge passes through a circuit element. The relationships for the voltage drops across each circuit element are summarized in Table 6.1.

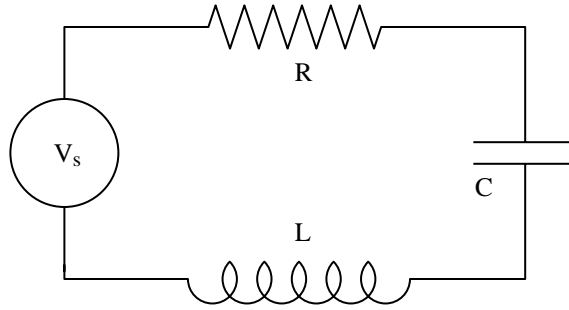
Imagine an electrical circuit with an alternating voltage source  $V_s(t)$  attached in series to a resistor, inductor, and capacitor (see Figure 6.4). The corresponding loop equation is

$$V_L + V_R + V_C = V_s(t). \quad (6.26)$$

The voltage source term  $V_s$  in (6.26) is the *emf* and is measured in units of volts. If we substitute the relationships shown in Table 6.1, we find

$$L \frac{d^2 Q}{dt^2} + R \frac{dQ}{dt} + \frac{Q}{C} = V_s(t), \quad (6.27)$$

where we have used the definition of current  $I = dQ/dt$ . We see that (6.27) for the series RLC circuit is identical in form to the damped harmonic oscillator (6.22). The analogies between ideal electrical circuits and mechanical systems are summarized in Table 6.2.

Figure 6.4: A simple series RLC circuit with a voltage source  $V_s$ .

Electric circuit	Mechanical system
charge $Q$	displacement $x$
current $I = dQ/dt$	velocity $v = dx/dt$
voltage drop	force
inductance $L$	mass $m$
inverse capacitance $1/C$	spring constant $k$
resistance $R$	damping $\gamma$

Table 6.2: Analogies between electrical parameters and mechanical parameters.

Although we are already familiar with (6.27), we first consider the dynamical behavior of an RC circuit described by

$$R \frac{dQ}{dt} = RI(t) = V_s(t) - \frac{Q}{C}. \quad (6.28)$$

Two RC circuits corresponding to (6.28) are shown in Figure 6.5. Although the loop equation (6.28) is identical regardless of the order of placement of the capacitor and resistor in Figure 6.5, the output voltage measured by the oscilloscope in Figure 6.5 is different. We will see in Problem 6.13 that these circuits act as *filters* that pass voltage components of certain frequencies while rejecting others.

An advantage of a computer simulation of an electrical circuit is that the measurement of a voltage drop across a circuit element does not affect the properties of the circuit. In fact, digital computers often are used to optimize the design of circuits for special applications. The `RCApp` program simulates an RC circuit with an alternating current (ac) voltage source of the form  $V_s(t) = \cos \omega t$  and shows the time dependencies of the voltage across the capacitor in a plotting panel. You are asked to modify this program in Problem 6.13.

*Problem 6.13.* Simple filter circuits

1. Copy the `RCApp` and `RC` classes from the chapter 6 package to a working directory. Modify the `RCApp` program to plot the voltage across the resistor,  $V_R$ , and the voltage across the source,  $V_s$ , in addition to the voltage across the capacitor,  $V_C$ . Run this program with  $R = 1000 \Omega$

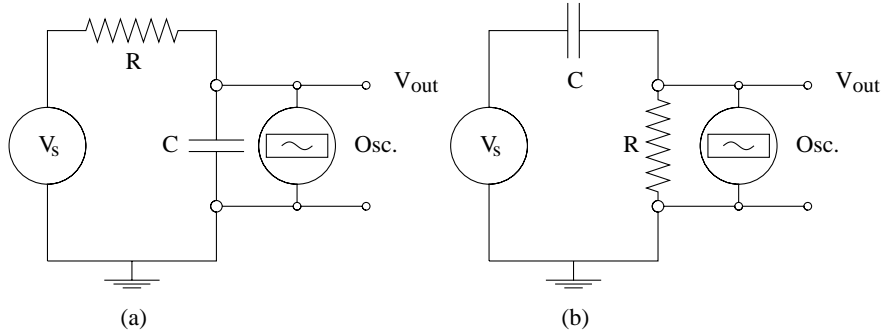


Figure 6.5: Examples of RC circuits used as low and high pass filters. Which circuit is which?

and  $C = 1.0 \mu F (10^{-6} \text{ farads})$ . Find the steady state amplitude of the voltage drops across the resistor and across the capacitor as a function of the angular frequency  $\omega$  of the source voltage  $V_s = \cos \omega t$ . Consider the frequencies  $f = 10, 50, 100, 160, 200, 500, 1000, 5000,$  and  $10000 \text{ Hz}$ . (Remember that  $\omega = 2\pi f$ .) Choose  $\Delta t$  to be no more than  $0.0001 \text{ s}$  for  $f = 10 \text{ Hz}$ . What is a reasonable value of  $\Delta t$  for  $f = 10000 \text{ Hz}$ ?

2. The output voltage depends on where the digital oscilloscope is connected. What is the output voltage of the oscilloscope in Figure 6.5a? Plot the ratio of the amplitude of the output voltage to the amplitude of the input voltage as a function of  $\omega$ . Use a logarithmic scale for  $\omega$ . What range of frequencies is passed? Does this circuit act as a high pass or a low pass filter? Answer the same questions for the oscilloscope in Figure 6.5b. Use your results to explain the operation of a high and low pass filter. Compute the value of the cutoff frequency for which the amplitude of the output voltage drops to  $1/\sqrt{2}$  (half-power) of the input value. How is the cutoff frequency related to  $RC$ ?
3. Plot the voltage drops across the capacitor and resistor as a function of time. The phase difference  $\phi$  between each voltage drop and the source voltage can be found by finding the time  $t_m$  between the corresponding maxima of the voltages. Because  $\phi$  is usually expressed in radians, we have the relation  $\phi/2\pi = t_m/T$ , where  $T$  is the period of the oscillation. What is the phase difference  $\phi_C$  between the capacitor and the voltage source and the phase difference  $\phi_R$  between the resistor and the voltage source? Do these phase differences depend on  $\omega$ ? Does the current lead or lag the voltage, that is, does the maxima of  $V_R(t)$  come before or after the maxima of  $V_s(t)$ ? What is the phase difference between the capacitor and the resistor? Does the latter difference depend on  $\omega$ ?
4. Modify the **RCApp** program to find the steady state response of an LR circuit with a source voltage  $V_s(t) = \cos \omega t$ . Let  $R = 100 \Omega$  and  $L = 2 \times 10^{-3} \text{ H}$ . Because  $L/R = 2 \times 10^{-5} \text{ s}$ , it is convenient to measure the time and frequency in units of  $T_0 = L/R$ . We write  $t^* = t/T_0$ ,  $\omega^* = \omega T_0$ , and rewrite the equation for an LR circuit as

$$I(t^*) + \frac{dI(t^*)}{dt^*} = \frac{1}{R} \cos \omega^* t^*. \tag{6.29}$$

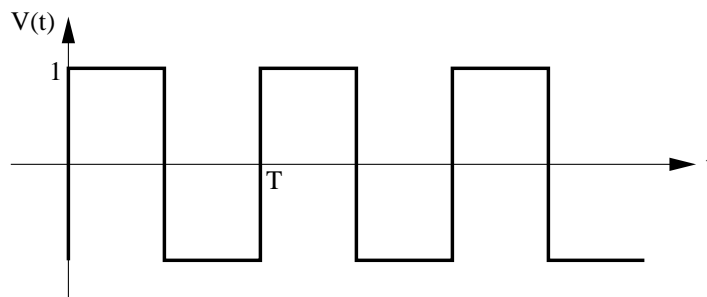


Figure 6.6: Square wave voltage with period  $T$  and unit amplitude.

Because it will be clear from the context, we now simply write  $t$  and  $\omega$  rather than  $t^*$  and  $\omega^*$ . What is a reasonable value of the step size  $\Delta t$ ? Compute the steady state amplitude of the voltage drops across the inductor and the resistor for the input frequencies  $f = 10, 20, 30, 35, 50, 100,$  and  $200$  Hz. Use these results to explain how an LR circuit can be used as a low pass or a high pass filter. Plot the voltage drops across the inductor and resistor as a function of time and determine the phase differences  $\phi_R$  and  $\phi_L$  between the resistor and the voltage source and the inductor and the voltage source. Do these phase differences depend on  $\omega$ ? Does the current lead or lag the voltage? What is the phase difference between the inductor and the resistor? Does the latter difference depend on  $\omega$ ?

*Problem 6.14.* Square wave response of an RC circuit

Modify the RCApp program so that the voltage source is a periodic square wave as shown in Figure 6.6. Use a  $1.0 \mu\text{F}$  capacitor and a  $3000 \Omega$  resistor. Plot the computed voltage drop across the capacitor as a function of time. Make sure the period of the square wave is long enough so that the capacitor is fully charged during one half-cycle. What is the approximate time dependence of  $V_C(t)$  while the capacitor is charging (discharging)?

We now consider the steady state behavior of the series RLC circuit shown in Figure 6.4 and represented by (6.27). The response of an electrical circuit is the current rather than the charge on the capacitor. Because we have simulated the analogous mechanical system, we already know much about the behavior of driven RLC circuits. Nonetheless, we will find several interesting features of ac electrical circuits in the following two problems.

*Problem 6.15.* Response of an RLC circuit

1. Consider an RLC series circuit with  $R = 100 \Omega$ ,  $C = 3.0 \mu\text{F}$ , and  $L = 2 \text{ mH}$ . Modify the simple harmonic oscillator program or the RC filter program to simulate an RLC circuit and compute the voltage drops across the three circuit elements. Assume an ac voltage source of the form  $V(t) = V_0 \cos \omega t$ . Plot the current  $I$  as a function of time and determine the maximum steady state current  $I_m$  for different values of  $\omega$ . Obtain the *resonance curve* by plotting  $I_m(\omega)$  as a function of  $\omega$  and compute the value of  $\omega$  at which the resonance curve is a maximum. This value of  $\omega$  is the *resonant frequency*.

2. The sharpness of the resonance curve of an ac circuit is related to the quality factor or  $Q$  value. ( $Q$  should not be confused with the charge on the capacitor.) The sharper the resonance, the larger the  $Q$ . Circuits with high  $Q$  (and hence a sharp resonance) are useful for tuning circuits in a radio so that only one station is heard at a time. We define  $Q = \omega_0/\Delta\omega$ , where the width  $\Delta\omega$  is the frequency interval between points on the resonance curve  $I_m(\omega)$  that are  $\sqrt{2}/2$  of  $I_m$  at its maximum. Compute  $Q$  for the values of  $R$ ,  $L$ , and  $C$  given in part (a). Change the value of  $R$  by 10% and compute the corresponding percentage change in  $Q$ . What is the corresponding change in  $Q$  if  $L$  or  $C$  is changed by 10%?
3. Compute the time dependence of the voltage drops across each circuit element for approximately fifteen frequencies ranging from 1/10 to 10 times the resonant frequency. Plot the time dependence of the voltage drops.
4. The ratio of the amplitude of the sinusoidal source voltage to the amplitude of the current is called the *impedance*  $Z$  of the circuit, that is,  $Z = V_m/I_m$ . This definition of  $Z$  is a generalization of the resistance that is defined by the relation  $V = IR$  for direct current circuits. Use the plots of part (c) to determine  $I_m$  and  $V_m$  for different frequencies and verify that the impedance is given by

$$Z(\omega) = \sqrt{R^2 + (\omega L - 1/\omega C)^2}. \quad (6.30)$$

For what value of  $\omega$  is  $Z$  a minimum? Note that the relation  $V = IZ$  holds only for the maximum values of  $I$  and  $V$  and not for  $I$  and  $V$  at any time.

5. Compute the phase difference  $\phi_R$  between the voltage drop across the resistor and the voltage source. Consider  $\omega \ll \omega_0$ ,  $\omega = \omega_0$ , and  $\omega \gg \omega_0$ . Does the current lead or lag the voltage in each case, that is, does the current reach a maxima before or after the voltage? Also compute the phase differences  $\phi_L$  and  $\phi_C$  and describe their dependence on  $\omega$ . Do the relative phase differences between  $V_C$ ,  $V_R$ , and  $V_L$  depend on  $\omega$ ?
6. Compute the amplitude of the voltage drops across the inductor and the capacitor at the resonant frequency. How do these voltage drops compare to the voltage drop across the resistor and to the source voltage? Also compare the relative phases of  $V_C$  and  $V_L$  at resonance. Explain how an RLC circuit can be used to amplify the input voltage.

## 6.9 Projects

*Project 6.16.* Chemical oscillations

The kinetics of chemical reactions can be modeled by a system of coupled first-order differential equations. As an example, consider the following reaction



where  $A, B$ , and  $C$  represent the concentrations of three different types of molecules. The corresponding rate equations for this reaction are

$$\frac{dA}{dt} = -kAB^2 \quad (6.32a)$$

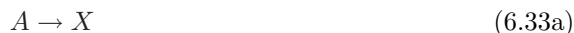
$$\frac{dB}{dt} = kAB^2 \quad (6.32b)$$

$$\frac{dC}{dt} = kAB^2. \quad (6.32c)$$

The rate at which the reaction proceeds is determined by the reaction constant  $k$ . The terms on the right-hand side of (6.32) are positive if the concentration of the molecule increases in (6.31) as it does for  $B$  and  $C$ , and negative if the concentration decreases as it does for  $A$ . Note that the term  $2B$  in the reaction (6.31) appears as  $B^2$  in the rate equation (6.32).

Most chemical reactions proceed to equilibrium, where the mean concentrations of all molecules are constant. However, if the concentrations of some molecules are replenished, it is possible to observe other kinds of behavior, such as oscillations (see below) and chaotic behavior (see Project ??). In (6.32) we have assumed that the reactants are well stirred, so that there are no spatial inhomogeneities. In Section ?? we will discuss the effects of spatial inhomogeneities due to molecular diffusion.

To obtain chemical oscillations, it is essential to have a series of chemical reactions such that the products of some reactions are the reactants of others. In the following, we consider a simple set of reactions that can lead to oscillations under certain conditions (see Lefever and Nicolis):



If we assume that the reverse reactions are negligible and  $A$  and  $B$  are held constant by an external source, the corresponding rate equations are

$$\frac{dX}{dt} = A - (B + 1)X + X^2Y \quad (6.34a)$$

$$\frac{dY}{dt} = BX - X^2Y. \quad (6.34b)$$

For simplicity, we have chosen the rate constants to be unity.

1. The steady state solution of (6.34) can be found by setting the left-hand side equal to zero. Show that the steady state values for  $(X, Y)$  are  $(A, B/A)$ .
2. Write a program to solve numerically the rate equations given by (6.34). A simple Euler algorithm is sufficient. Your program should input the initial values of  $X$  and  $Y$  and the fixed concentrations  $A$  and  $B$ , and plot  $X$  versus  $Y$  as the reactions evolve.
3. Systematically vary the initial values of  $X$  and  $Y$  for given values of  $A$  and  $B$ . Are their steady state behaviors independent of the initial conditions?

4. Let the initial value of  $(X, Y)$  equal  $(A + 0.001, B/A)$  for several different values of  $A$  and  $B$ , that is, choose initial values close to the steady state values. Classify which initial values result in steady state behavior (stable) and which ones show periodic behavior (unstable). Find the relation between  $A$  and  $B$  that separates the two types of behavior.

*Project 6.17.* Comparison of algorithms

1. Consider a particle of unit mass moving in the Morse potential:

$$V(x) = e^{-2x} - 2e^{-x}. \quad (6.35)$$

The total energy  $E = \frac{1}{2}p^2 + V(x) < 0$ , and the force on the particle is given by

$$F(x) = -\frac{dV}{dx} = 2e^{-x}(e^{-x} - 1). \quad (6.36)$$

Plot  $V(x)$  and  $F(x)$  versus  $x$ . What is their qualitative dependence on  $x$ ? What is the total energy for the initial condition  $x_0 = 2$  and  $v_0 = 0$ ? What type of motion do you expect?

2. Compare the Euler, Euler-Richardson, and Verlet algorithms by computing  $x(t)$ ,  $v(t)$ , and  $E_n$ , where  $E_n$  is the total energy after the  $n$ th step. One measure of the error associated with the algorithm is  $\Delta E_{\max}$ , the maximum value of the difference  $|E_n - E_0|$  over the time interval  $t_n - t_0$ . Compute this measure of the error at a given value of  $t_n$  for decreasing values of  $\Delta t$  with the initial condition  $(x_0 = 2, v_0 = 0)$ . If an algorithm is second-order, the error in the total energy should decrease as  $(\Delta t)^2$ . The absence of such a decrease might indicate that there is an error in your program or that roundoff errors are important. Which one of the above algorithms is the best trade-off between speed, accuracy, and simplicity?
3. Compare your results for  $x(t)$  to the analytical result

$$x(t) = \ln(\alpha \cos \omega t + \beta \sin \omega t - E_0^{-1}) \quad (6.37)$$

with

$$\alpha = e^{x_0} + E_0^{-1} \quad (6.38)$$

$$\beta = \omega^{-1} v_0 e^{x_0} \quad (6.39)$$

$$\omega = (2|E_0|)^{1/2}. \quad (6.40)$$

4. Repeat the computations of part (a) with  $x_0 = 3, v_0 = 1$ . Is the motion periodic? Which algorithm is most suitable in this case?

## References and Suggestions for Further Reading

- F. S. Acton, *Numerical Methods That Work*, The Mathematical Association of America (1990), Chapter 5.

- G. L. Baker and J. P. Gollub, *Chaotic Dynamics: An Introduction*, Cambridge University Press (1990). A good introduction to the notion of phase space.
- A. Douglas Davis, *Classical Mechanics*, Academic Press (1986). The author gives a “crash” course in conversational BASIC and Pascal and simple numerical solutions of Newton’s equations of motion. Much emphasis is given to the harmonic oscillator problem.
- S. Eubank, W. Miner, T. Tajima, and J. Wiley, “Interactive computer simulation and analysis of Newtonian dynamics,” *Amer. J. Phys.* **57**, 457 (1989).
- Richard P. Feynman, Robert B. Leighton, and Matthew Sands, *The Feynman Lectures on Physics*, Vol. 1, Addison-Wesley (1963). Chapters 21, 23, 24, and 25 are devoted to various aspects of harmonic motion.
- Charles Kittel, Walter D. Knight, and Malvin A. Ruderman, *Mechanics*, second edition, revised by A. Carl Helmholz and Burton J. Moyer, McGraw-Hill (1973).
- R. Lefever and G. Nicolis, “Chemical instabilities and sustained oscillations,” *J. Theor. Biol.* **30**, 267 (1971).
- Jerry B. Marion and Stephen T. Thornton, *Classical Dynamics*, fourth edition, Academic Press (1995). Excellent discussion of linear and nonlinear oscillators.
- M. F. McInerney, “Computer-aided experiments with the damped harmonic oscillator,” *Am. J. Phys.* **53**, 991 (1985).
- William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes*, second edition, Cambridge University Press (1992). Chapter 16 discusses the integration of ordinary differential equations.
- J. C. Sprott, *Introduction to Modern Electronics*, John Wiley & Sons (1981). The first five chapters treat the topics discussed in Section 6.8.
- S. C. Zilio, “Measurement and analysis of large-angle pendulum motion,” *Am. J. Phys.* **50**, 450 (1982).

There are many good books on Java drawing and Java threads. We list a few of our favorites:

- David M. Geary, *Graphic Java: Mastering the JFC, Vol. 1 AWT and Vol. 2 Swing*, Prentice Hall (1999).
- Jonathan Knudsen, *Java 2D Graphics*, O’Reilly (1999).
- Scott Oaks and Henry Wong, *Java Threads*, second edition, O’Reilly (1999).