differential equation. A simple model of population growth that relates the population in generation $n+1$ to the population in generation n is given by

$$P_{n+1} = aP_n, \tag{6.2}$$

where P_n is the population in generation n and a is a constant. In the following, we will assume that the time interval between generations is unity and will refer to n as the time.

If $a < 1$, the population decreases at each generation, and eventually the population becomes extinct. If $a > 1$, each generation will be a times larger than the previous one. In this case (6.2) leads to geometrical growth and an unbounded population. Although the unbounded nature of geometrical growth is clear, it is remarkable that most of us do not integrate our understanding of geometrical growth into our everyday lives. Can a bank pay 4% interest each year indefinitely? Can the world's human population grow at a constant rate forever?

It is natural to formulate a more realistic model in which the population is bounded by the finite carrying capacity of its environment. A simple model of density-dependent growth is

$$P_{n+1} = P_n(a - bP_n). \tag{6.3}$$

Equation (6.3) is nonlinear due to the presence of the quadratic term in P_n. The linear term represents the natural growth of the population; the quadratic term represents a reduction of this natural growth caused, for example, by overcrowding or by the spread of disease.

It is convenient to rescale the population by letting $P_n = (a/b)x_n$ and rewriting (6.3) as

$$x_{n+1} = ax_n(1 - x_n). \tag{6.4}$$

The replacement of P_n by x_n changes the units used to define the various parameters. To write (6.4) in the standard form (6.1), we define the parameter $r = a/4$ and obtain

$$x_{n+1} = f(x_n) = 4rx_n(1 - x_n). \tag{6.5}$$

The rescaled form (6.5) has the desirable feature that its dynamics are determined by a single control parameter r instead of the two parameters a and b. Note that if $x_n > 1$, x_{n+1} will be negative. To avoid this unphysical feature, we impose the conditions that x and r are restricted to the intervals $0 \le x \le 1$ and $0 < r \le 1$, respectively. Because the function $f(x)$ defined in (6.5) transforms any point on the one-dimensional interval $[0, 1]$ into another point in the same interval, the function f is called a *one-dimensional map*.

The form of $f(x)$ in (6.5) is known as the *logistic* map. The logistic map is a simple example of a *dynamical system*; that is, the map is a deterministic, mathematical prescription for finding the future state of a system given its present state.

The sequence of values $x_0, x_1, x_2, \ldots$ is called the *trajectory*. To check your understanding, suppose that the initial value of x_0 or *seed* is $x_0 = 0.5$ and $r = 0.2$. Do a calculation to show that the trajectory is $x_1 = 0.2, x_2 = 0.128, x_3 = 0.089293, \ldots$. The first thirty iterations of (6.5) are shown for two values of r in Figure 6.1.

The class `IterateMapApp` computes the trajectory of the logistic map in (6.5). Note that we have extended the `AbstractCalculation` class, which is appropriate because many of the results of Sections 6.1–6.4 were discovered using a programmable calculator.

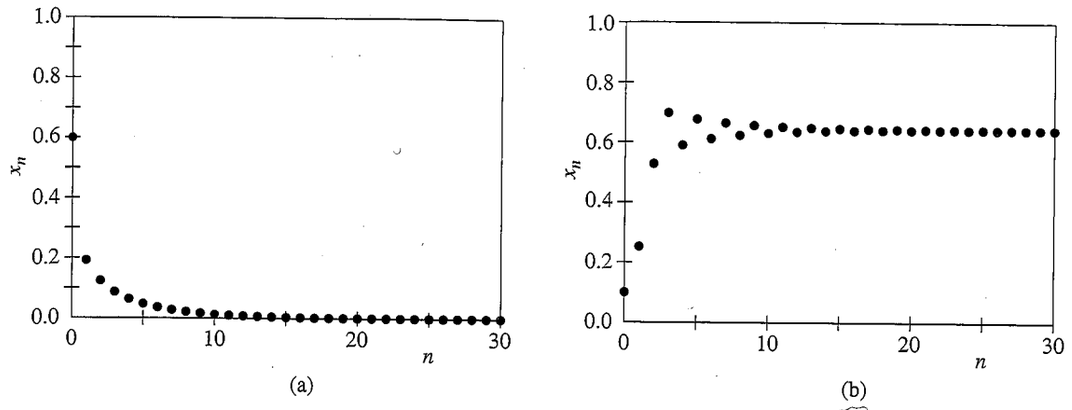

Figure 6.1 (a) The trajectory of x for $r = 0.2$ and $x_0 = 0.6$. The stable fixed point is at $x = 0$. (b) The trajectory for $r = 0.7$ and $x_0 = 0.1$. Note the initial transient behavior.

Listing 6.1 The `IterateMapApp` class iterates the logistic map and plots the resulting trajectory.

```
package org.opensourcephysics.sip.ch06;
import org.opensourcephysics.frames.*;
import org.opensourcephysics.controls.*;

public class IterateMapApp extends AbstractCalculation {
   int datasetIndex = 0;
   PlotFrame plotFrame = new PlotFrame("iterations", "x",
         "trajectory");
   public IterateMapApp() {
      // keep data between calls to calculate
      plotFrame.setAutoclear(false);
   }

   public void reset() {
      control.setValue("r", 0.2);
      control.setValue("x", 0.6);
      control.setValue("iterations", 50);
      datasetIndex = 0;
   }

   public void calculate() {
      double r = control.getDouble("r");
      double x = control.getDouble("x");
      int iterations = control.getInt("iterations");
      for(int i = 0;i<=iterations;i++) {
         plotFrame.append(datasetIndex, i, x);
         x = map(r, x);
      }
      plotFrame.setMarkerSize(datasetIndex, 1);
      plotFrame.setXYColumnNames(datasetIndex, "iteration",
            "calc #"+datasetIndex);
      datasetIndex++;
   }
```