the trajectory falls into each bin. At each value of r in the range $0.7 \leq r \leq 1$, the map should be iterated for a fixed number of steps, for example, $n = 1000$. Choose $\Delta x = 0.01$. What happens to the entropy when the trajectory is chaotic?

(b) Repeat part (a) with $n = 10{,}000$. For what values of r does the entropy change significantly? Decrease Δx to 0.001 and repeat. Does this decrease make a difference?

(c) Plot p_i as a function of x for $r = 1$. For what value(s) of x is the plot a maximum? ■

We can also measure the (generalized) entropy as a function of time. As we will see in Problem 6.12, $S(n)$ for strong chaos increases linearly with n until all the possible states are visited. However, for weak chaos this behavior is not found. In the latter case we can generalize the entropy to a q-dependent function defined by

$$S_q = \frac{1 - \sum_i p_i^q}{q - 1}. \tag{6.27}$$

In the limit $q \to 1$, $S_q \to S$. The following problem discusses measuring the entropy for the same system as in Problem 6.10.

***Problem 6.12 Entropy of weak and strong chaotic systems**

(a) Write a program that iterates the map (6.24) and plots S, if $q = 1$, or S_q, if $q \neq 1$, as a function of n. The input parameters should be q, the number of bins, the number of random seeds in a single bin, and n, the number of iterations. At each iteration compute the entropy. Then average S over the randomly chosen values of the seeds.

(b) Consider strong chaos at $a = 2$. Choose $q = 1$, $n = 20$, $\Delta x \leq 0.001$, and ten randomly chosen seeds per bin. Do you obtain a straight line for S versus n? Does the curve eventually stop growing? If you decrease Δx, how do your results differ and how are they the same? Show that S is not a linear function of n if $q \neq 1$.

(c) Repeat part (a) with $a = 1.401155189$ and various values of q. Simulations with 10^5 bins show that linear behavior is obtained for $q \approx 0.36$, the same value as for the measurements in Problem 6.10. ■

*6.6 ■ CONTROLLING CHAOS

The dream of classical physics was that if the initial conditions and all the forces acting on a system were known, then we could predict the future with as much precision as we desire.

The existence of chaos has shattered that dream. However, if a system is chaotic, we can still control its behavior with small, but carefully chosen, perturbations of the system. We will illustrate the method for the logistic map. The application of the method to other one-dimensional systems is straightforward, but the extension to higher-dimensional systems is more complicated (cf. Ott, Lai).

Suppose that we want the trajectory to be periodic even though the parameter r is in the chaotic regime. How can we make the trajectory have periodic behavior without drastically changing r or imposing an external perturbation that is so large that the internal dynamics

of the map become irrelevant? The key idea is that for any value of r in the chaotic regime, there is an infinite number of trajectories that have unstable periods. This property of the chaotic regime means that if we choose the value of the seed x_0 to be precisely equal to a point on an unstable trajectory with period p, the subsequent trajectory will have this period. However, if we choose a value of x_0 that differs ever so slightly from this special value, the trajectory will not be periodic. Our goal is to make slight perturbations to the system to keep it on the desired unstable periodic trajectory.

The first step is to find the values of $x(i)$, $i = 1$ to p, that constitute the unstable periodic trajectory. It is an interesting numerical problem to find the values of $x(i)$, and we consider this problem first. To find a fixed point of the map $f^{(p)}$, we need to find the value of x^* such that

$$g^{(p)}(x^*) \equiv f^{(p)}(x^*) - x^* = 0. \tag{6.28}$$

The algorithms for finding the solution to (6.28) are called *root-finding* algorithms. You might have heard of Newton's method, which we describe in Appendix 6B. Here we use the simplest root-finding algorithm, the *bisection* method. The algorithm works as follows:

1. Choose $x_{\rm left}$ and $x_{\rm right}$, with $x_{\rm left} < x_{\rm right}$, such that $g^{(p)}(x_{\rm left})g^{(p)}(x_{\rm right}) < 0$. Because this product is negative, there must be a value of x such that $g^{(p)}(x) = 0$ in the interval $[x_{\rm left}, x_{\rm right}]$.
2. Choose the midpoint, $x_{\rm mid} = x_{\rm left} + \frac{1}{2}(x_{\rm right} - x_{\rm left}) = \frac{1}{2}(x_{\rm left} + x_{\rm right})$, as the guess for x^*.
3. If $g^{(p)}(x_{\rm mid})$ has the same sign as $g^{(p)}(x_{\rm left})$, then replace $x_{\rm left}$ by $x_{\rm mid}$; otherwise, replace $x_{\rm right}$ by $x_{\rm mid}$. The interval for the location of the root is now reduced.
4. Repeat steps 2 and 3 until the desired level of precision is achieved.

The following program implements this algorithm for the logistic map. An alternative implementation named `FixedPointApp` that does not use recursion is not listed, but is available in the ch06 package. One possible problem is that some of the roots of $g^{(p)}(x) = 0$ are also roots of $g^{(p')}(x) = 0$ for p' equal to a factor of p. (For example, if $p = 6$, 2 and 3 are factors.) As p increases, it might become more difficult to find a root that is part of a period p trajectory and not part of a period p' trajectory.

Listing 6.4 The `RecursiveFixedPointApp` program finds stable and unstable periodic trajectories with the given period using the bisection root-finding algorithm.

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

public class RecursiveFixedPointApp extends AbstractCalculation {
   double r;
   int period;
   double xleft, xright;
   double gleft, gright;

   public void reset() {
      control.setValue("r", 0.8);           // control parameter r
      control.setValue("period", 2);        // period
```