Invariant measures

(Sethna, "Entropy, Order Parameters, and Complexity", ex. 4.3)

Liouville's theorem tells us that all available points in phase space are equally weighted when a Hamiltonian system is averaged over all times. What happens for systems that evolve according to laws that are not Hamiltonian? Usually, the system does not continue to explore all points in its state space; at long times it is confined to a subset of the original space known as the *attractor*.

Import packages

In [ ]:

```
%pylab inline
from scipy import *
```

We consider the behavior of the `logistic' mapping from the unit interval $(0, 1)$ into itself:

$$f(x) = 4\mu x(1 - x).$$

(We also study this map in the exercises "Chaos, Lyapunov, and entropy increase", "Fractal Dimensions", and "Period doubling"). We talk of the trajectory of an initial point x_0 as the sequence of points $x_0, f(x_0), f(f(x_0))$, \dots, $f^{[n]}(x_0)$, ... Iteration can be thought of as a time step (one iteration of a Poincare return map of the exercise "Jupiter" or one step Δt in a time-step algorithm.

In [ ]:

```
def f(x,mu):
    """
    Logistic map f(x) = 4 mu x (1-x), which folds the unit interval (0,1)
    into itself.
    """
    return 4*...
```

Attracting fixed point. For small μ, our mapping has an attracting fixed-point. A fixed-point of a mapping is a value $x^* = f(x^*)$; a fixed-point is stable if small perturbations shrink after iterating: \begin{equation} |f(x^ + \epsilon) - x^| \approx |f'(x^)| \epsilon < \epsilon, \end{equation} which happens if the derivative $|f'(x^)|<1$. ($For many-dimensional mappings, a sufficient criterion for stability is that all the eigenvalues of t$. A continuous time evolution $\dot{x} = F(y)$ will be stable if dF/dy is smaller than zero, or (for multidimensional systems) if the Jacobian DF has eigenvalues whose real parts are all less than zero.)

(a) **Iteration.** *Set $\mu = 0.2$; iterate f for some initial points $0*

Analytics. *Find the non-zero fixed-point $x^*(\mu)$ of the map $f(x)$, and show that it exists and is stable for $1/4 < \mu < 3/4$. If you are ambitious or have a computer algebra program, show that there is a stable, period-two cycle for $3/4 < \mu < (1 + \sqrt{6})/4$.*

In [ ]:

```
x = 0.77
for t in range(100):
    x = ...(...,0.2)
print("x goes to", x, "when f(x,0.2) is iterated")

figure()
xs = arange(0,1,0.01)
plot(xs, xs)
for mu in (0.2, 0.4, 0.6):
    plot(xs, f(...), label = "mu=%g" %mu)
legend(loc=2)
axes().set_aspect('equal')
```

An attracting fixed-point is the antithesis of Liouville's theorem; all initial conditions are transient except one, and all systems lead eventually to the same, time-independent state. (On the other hand, this is precisely the behavior we expect in statistical mechanics on the macroscopic scale; the system settles down into a time-independent equilibrium state! All microstates are equivalent, but the vast majority of accessible microstates have the same macroscopic behavior in most large systems.) We could define a rather trivial `equilibrium ensemble' for this system, which consists of the single point x^*; any property $O(x)$ will have the long-time average $\langle O \rangle = O(x^*)$.

For larger values of μ, more complicated things happen. At $\mu = 1$, the dynamics can be shown to fill the entire interval; the dynamics is ergodic, and the attractor fills the entire set of available states. However, unlike the case of Hamiltonian systems, not all states are weighted equally (i.e., Liouville's theorem does not hold).

We can find time averages for functions of x in two ways: by averaging over time (many iterates of the map) or by weighting an integral over x by the invariant density $\rho(x)$. The invariant density $\rho(x)\,dx$ is the probability that a point on a long trajectory will lie between x and $x + dx$. To find it numerically, we iterate a typical point (not an unstable fixed-point or unstable periodic orbit!) x_0 a thousand or so times ($N_{\text{transient}}$) to find a point x_a on the attractor, and then collect a long trajectory of perhaps a million points (N_{cycles}). A histogram of this trajectory gives $\rho(x)$. Averaging over this density is manifestly the same as a time average over the trajectory of a million points. We call $\rho(x)$ invariant because it is left the same under the mapping f; iterating our million-point approximation for ρ once under f only removes the first point x_a and adds one extra point to the end.

(b) ***Invariant density.*** *Set $\mu = 1$; iterate f many times, and form a histogram of values giving the density $\rho(x)$ of points along the trajectory. You should find that points x near the boundaries are approached more often than points near the center.*
Analytics. *Using the fact that the long-time average $\rho(x)$ must be independent of time, verify for $\mu = 1$ that the density of points is*

$$\rho(x) = \frac{1}{\pi\sqrt{x(1-x)}}.$$

(You need not derive the factor $1/\pi$, which normalizes the probability density to one.) Plot this theoretical curve with your numerical histogram. (Hint: The points in a range dx around a point x map under f to a range $dy = f'(x)\,dx$ around the image $y = f(x)$. Each iteration maps two points x_a and $x_b = 1 - x_a$ to y, and thus maps all the density $\rho(x_a)|dx_a|$ and $\rho(x_b)|dx_b|$ into dy. Hence the probability $\rho(y)\,dy$ must equal $\rho(x_a)|dx_a| + \rho(x_b)|dx_b|$, so

$$\rho(f(x_a)) = \rho(x_a)/|f'(x_a)| + \rho(x_b)/|f'(x_b)|.$$

Substitute the second-to-latest equation for $\rho(x)$ above into this equation. You will need to factor a polynomial.)

In [ ]:

```
def IterateList(x,mu,Niter=10,Nskip=0):
    """
    Iterate the function f(x,mu) Niter-1 times, starting at x
    (or at x iterated Nskip times), so that the full trajectory
    contains N points.
    Returns the entire list
    (x, f(x), f(f(x)), ... f(f(...(f(x))...))).

    Can be used to explore the dynamics starting from an arbitrary point
    x0, or to explore the attractor starting from a point x0 on the
    attractor (say, initialized using Nskip).

    For example, you can use Iterate to find a point xAttractor on the
    attractor and IterateList to create a long series of points attractorX
s
    (thousands, or even millions long, if you're in the chaotic region),
    and then use
        hist(attractorXs, bins=2000, normed=1)
    to see the density of points.
    """
    for i in range(Nskip):
        x = f(x,mu)
    fiter = [x]
    for i in range(Niter-1):
        x = ...
        fiter.append(x)
    return fiter

hist(IterateList(..., 1.0, Niter=1000000, Nskip=1000), bins = 2000, normed
 = 1);

xs = arange(0.001,0.999,0.001)
plot(xs, ..., "r-");
```

Mathematicians call this probability density $\rho(x)\,dx$ the invariant measure on the attractor. (There are actually many possible invariant measures on some attractors; this one is the SRB measure (John Guckenheimer, private communication).) To get the long-term average of any function $O(x)$, one can use

$$\langle O \rangle = \int O(x)\rho(x)\,dx.$$

To a mathematician, a measure is a way of weighting different regions when calculating integrals---precisely our $\rho(x)\,dx$. Notice that, for the case of an attracting fixed-point, we would have $\rho(x) = \delta(x^*)$. (The case of a fixed-point then becomes mathematically a measure with a point mass at x^*.)

Cusps in the invariant density. *At values of μ slightly smaller than one, our mapping has a rather complex invariant density.*

(c) Find the invariant density (as described above) for $\mu = 0.9$. Make your trajectory length N_{cycles} big enough and the bin size small enough to see the interesting structures. Notice that the attractor no longer fills the whole range $(0, 1)$; locate roughly where the edges are. Notice also the cusps in $\rho(x)$ at the edges of the attractor, and also at places inside the attractor (called 'boundaries'). Locate some of the more prominent cusps.

In [ ]:

```
hist(IterateList(..., 0.9, Niter=1000000, Nskip=1000), bins = 2000, normed
 = 1);
```

Analytics of cusps. *Notice that $f'(1/2) = 0$, so by the equation for $\rho(f(x_a))$ above we know that $\rho(f(x)) \geq \rho(x)/|f'(x)|$ must have a singularity near $x = 1/2$; all the points near $x = 1/2$ are squeezed together and folded to one side by f. Further iterates of this singularity produce more cusps; the crease after one fold stays a crease after being further stretched and kneaded.*

(d) Set $\mu = 0.9$. Calculate $f(1/2)$, $f(f(1/2))$, ... and compare these iterates to the locations of the edges and cusps from part~(c). (You may wish to include them both on the same plot.)

In [ ]:

```
hist(...);

boundaries = IterateList(...,0.9,8,1)
plot(boundaries, 1.+arange(len(boundaries)), 'ro-')
```

Bifurcation diagram. *The evolution of the attractor and its invariant density as μ varies are plotted in the bifurcation diagram. One of the striking features in this plot are the sharp boundaries formed by the cusps.*

(e) ***Bifurcation diagram.*** *Plot the attractor as a function of μ, for $0.9 < \mu < 1$. (Pick regularly-spaced $\delta\mu$, run $n_{\text{transient}}$ steps, record n_{cycles} steps, and plot. After the routine is working, you should be able to push $n_{\text{transient}}$ and n_{cycles} both larger than 100, and $\delta\mu < 0.01$.)*
On the same plot, for the same μs, plot the first eight images of $x = 1/2$, that is, $f(1/2), f(f(1/2)), \ldots$. Are the boundaries you see just the cusps? What happens in the bifurcation diagram when two boundaries touch? </EM>

In [ ]:

```
def BifurcationDiagram(muMin=0., muMax=1., deltaMu=0.0002, x0=0.49,
Nskip=100,
                       Niter=512, marker = '.', size=1, color='b'):
    xlim((muMin,muMax))
    muArray = arange(muMin, muMax, deltaMu)
    mus = []
    trajs = []
    for mu in muArray:
        mus.extend([mu]*Niter)
        trajs.extend(IterateList(x0, mu, Niter, Nskip))
    scatter(mus, trajs, marker = marker, s=size,c=color)
    setp(axes().get_xticklabels(),fontsize=40)
    setp(axes().get_yticklabels(),fontsize=40)
    ylabel('x', fontsize=40)
    xlabel('mu', fontsize=40)
    return fig

figure(figsize=(50,20))
BifurcationDiagram(0.9,1.0)
BifurcationDiagram(0.9,1.0,x0=...,Nskip=...,Niter=8,marker='o',color='red',

                           size=30);
```