Fractal dimensions

(Sethna, "Entropy, Order Parameters, and Complexity", ex. 5.16)

There are many strange sets that emerge in science. In statistical mechanics, such sets often arise at continuous phase transitions, where self-similar spatial structures arise (Chapter 12). In chaotic dynamical systems, the attractor (the set of points occupied at long times after the transients have disappeared) is often a fractal (called a *strange attractor*). These sets are often tenuous and jagged, with holes on all length scales, as in percolation (Fig. 1.2).

We often try to characterize these strange sets by a dimension. The dimensions of two extremely different sets can be the same; the path exhibited by a random walk (embedded in three or more dimensions) is arguably a two-dimensional set, but does not locally look like a surface. However, if two sets have different spatial dimensions (measured in the same way) they are certainly qualitatively different.

There is more than one way to define a dimension. Roughly speaking, strange sets are often spatially inhomogeneous, and what dimension you measure depends upon how you weight different regions of the set. In this exercise, we will calculate the *information dimension (closely connected to the non-equilibrium entropy), and the capacity dimension (originally called the Hausdorff dimension}, also sometimes called the fractal dimension).*

Import packages

In [ ]:
```
# Sometimes gives interactive new windows
# Must show() after plot, figure() before new plot
# %matplotlib

# Adds static figures to notebook: good for printing
%matplotlib inline

# Interactive windows inside notebook! Must include plt.figure() betwee
# %matplotlib notebook

# Better than from numpy import *, but need np.sin(), np.array(), plt.p
import numpy as np
import matplotlib.pyplot as plt
```

To generate our strange set---along with some more ordinary sets---we will use the logistic map

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

The attractor for the logistic map is a periodic orbit (dimension zero) at $\mu = 0.8$, and a chaotic, cusped density filling two intervals (dimension one) at $\mu = 0.9$. (See the exercise 'Invariant measures'. The chaotic region for the logistic map does not have a strange attractor because the map is confined to one dimension; period-doubling cascades for dynamical

systems in higher spatial dimensions have fractal, strange attractors in the chaotic region. At the onset of chaos at $\mu = \mu_\infty \approx 0.892486418$ (see the exercise 'Period doubling') the dimension becomes intermediate between zero and one; this strange, self-similar set is called the *Feigenbaum attractor.*

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

Both the information dimension and the capacity dimension are defined in terms of the occupation P_n of cells of size ϵ in the limit as $\epsilon \to 0$.

(a) *Write a routine which, given μ and a set of bin sizes ϵ, does the following:*

- Iterates f hundreds or thousands of times (to get onto the attractor).
- Iterates f a large number $N_{\rm tot}$ more times, collecting points on the attractor. (For $\mu \leq \mu_\infty$, you could just integrate 2^n times for n fairly large.)
- For each ϵ, use a histogram to calculate the probability P_j that the points fall in the jth bin.
- Return the set of vectors $P_j[\epsilon]$. You may wish to test your routine by using it for $\mu = 1$ (where the distribution should look like $\rho(x) = 1/\pi\sqrt{x(1-x)}$, see the exercise 'Invariant measures') and $\mu = 0.8$ (where the distribution should look like two δ-functions, each with half of the points).

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 poir
    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 attract
    (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

def GetPn(mu, epsilonList, Niter, Nskip=10000):
    """
    Generates probability arrays P_n[epsilon].
    Specifically,
     finds a point on the attractor by iterating Nskip times
     collects points on the attractor of size Niter
     for each epsilon in epsilonList,
      generates bins of size epsilon for the range (0,1) of the functic
          bins = np.arange(0.0,1.0+eps,eps)
      finds the number of points from the sample in each bin, using
      the histogram function
          numbers, bins = np.histogram(sample, bins=bins)
      and computes the probability P_n[epsilon] of being in each bin.
    In the period doubling region the sample should of size 2^n so that
    it covers the attractor evenly.
    """
    sample = IterateList(0.1, mu, Niter, Nskip)
    P_n = {}
    for eps in epsilonList:
        bins = np.arange(0.0, 1.0 + eps, eps)
        numbers, bins = np.histogram(sample, bins=bins)
        P_n[eps] = ...  # Probability
    return P_n

Pn = GetPn(0.8,[0.001],10000)
plt.plot(Pn[0.001])
plt.figure()
Pn = GetPn(1.0,[0.001],10000)
```

```
plt.plot(Pn[0.001])
```

The capacity dimension. The definition of the capacity dimension is motivated by the idea that it takes at least

$$N_{\text{cover}} = V/\epsilon^D$$

bins of size ϵ^D to cover a D-dimensional set of volume V. (Imagine covering the surface of a sphere in 3D with tiny cubes; the number of cubes will go as the surface area (2D volume) divided by ϵ^2.) By taking logs of both sides we find $\log N_{\text{cover}} \approx \log V + D \log \epsilon$. The capacity dimension is defined as the limit

$$D_{\text{capacity}} = \lim_{\epsilon \to 0} \frac{\log N_{\text{cover}}}{\log \epsilon},$$

but the convergence is slow (the error goes roughly as $\log V / \log \epsilon$). Faster convergence is given by calculating the slope of $\log N$ versus $\log \epsilon$:

$$\begin{aligned} D_{\text{capacity}} &= \lim_{\epsilon \to 0} \frac{d \log N_{\text{cover}}}{d \log \epsilon} \\ &= \lim_{\epsilon \to 0} \frac{\log N_{j+1} - \log N_j}{\log \epsilon_{i+1} - \log \epsilon_i}. \end{aligned}$$

(b) *Use your routine from part (a), write a routine to calculate $N[\epsilon]$ by counting non-empty bins. Plot D_{capacity} from the fast convergence versus the midpoint $(1/2)(\log \epsilon_{i+1} + \log \epsilon_i)$. Does it appear to extrapolate to $D = 1$ for $\mu = 0.9$?,% (In the chaotic regions, keep the number of bins small compared to the number of iterates in your sample, or you will start finding empty bins between points and eventually get a dimension of zero.) Does it appear to extrapolate to $D = 0$ for $\mu = 0.8$? Plot these two curves together with the curve for μ_∞. Does the last one appear to converge to $D_1 \approx 0.538$, the capacity dimension for the Feigenbaum attractor gleaned from the literature? How small a deviation from μ_∞ does it take to see the numerical cross-over to integer dimensions?*

Entropy and the information dimension. The probability density $\rho(x_j) \approx P_j/\epsilon = (1/\epsilon)(N_j/N_{\text{tot}})$. Converting the non-equilibrium entropy formula to a sum gives

$$\begin{aligned} S &= -k_B \int \rho(x) \log(\rho(x))\, dx \\ &\approx -\sum_j \frac{P_j}{\epsilon} \log\left(\frac{P_j}{\epsilon}\right) \epsilon \\ &= -\sum_j P_j \log P_j + \log \epsilon \end{aligned}$$

(setting the conversion factor $k_B = 1$ for convenience).

You might imagine that the entropy for a fixed-point would be zero, and the entropy for a period-m cycle would be $k_B \log m$. But this is incorrect; when there is a fixed-point or a periodic limit cycle, the attractor is on a set of dimension zero (a bunch of points) rather than dimension one. The entropy must go to minus infinity---since we have precise information about where the trajectory sits at long times. To estimate the 'zero-dimensional' entropy $k_B \log m$ on the computer, we would use the discrete form of the entropy summing over bins P_j instead of integrating over x:

$$S_{d=0} = -\sum_j P_j \log(P_j) = S_{d=1} - \log(\epsilon).$$

More generally, the 'natural' measure of the entropy for a set with D dimensions might be defined as

$$S_D = -\sum_j P_j \log(P_j) + D \log(\epsilon).$$

Instead of using this formula to define the entropy, mathematicians use it to define the information dimension

$$D_{\text{inf}} = \lim_{\epsilon \to 0} \left(\sum P_j \log P_j \right) / \log(\epsilon).$$

The information dimension agrees with the ordinary dimension for sets that locally look like $\mathbb{R}^D$. It is different from the capacity dimension, which counts each occupied bin equally; the information dimension counts heavily occupied parts (bins) in the attractor more heavily. Again, we can speed up the convergence by noting that the equation for the information dimension says that $\sum_j P_j \log P_j$ is a linear function of $\log \epsilon$ with slope D and intercept S_D. Measuring the slope directly, we find

$$D_{\text{inf}} = \lim_{\epsilon \to 0} \frac{d \sum_j P_j(\epsilon) \log P_j(\epsilon)}{d \log \epsilon}.$$

(c) *As in part (b), write a routine that plots D_{inf} using the fast definition as a function of the midpoint $\log \epsilon$, as we increase the number of bins. Plot the curves for $\mu = 0.9$, $\mu = 0.8$, and μ_∞. Does the information dimension agree with the ordinary one for the first two? Does the last one appear to converge to $D_1 \approx 0.517098$, the information dimension for the*

In [ ]:

```
def DimensionEstimates(mu, epsilonList, Niter):
    """
    Estimates the capacity dimension and the information dimension
    for a sample of points on the line.
    The capacity dimension is defined as
       D_capacity = lim {eps->0} (- log(Nboxes) / log(eps))
    but converges faster as
       D_capacity = - (log(Nboxes[i+1])-log(Nboxes[i]))
                        / (log(eps[i+1])-log(eps[i]))
    where Nboxes is the number of intervals of size eps needed to
    cover the space. The information dimension is defined as
       D_inf = lim {eps->0} sum(P_n log P_n) / log(eps)
    but converges faster as
       S0 = sum(P_n log P_n)
       D_inf = - (S0[i+1]-S0[i])
                        / (log(eps[i+1])-log(eps[i]))
    where P_n is the fraction of the list 'sample' that is in bin n,
    and the bins are of size epsilon. You'll need to add a small
    increment delta to P_n before taking the log: delta = 1.e-100 will
    not change any of the non-zero elements, and P_n log (P_n + delta)
    will be zero if P_n is zero.

    Returns three lists, with epsilonBar (geometric mean of neighboring
    epsilonList values), and D_inf, and D_capacity values for each
    epsilonBar
    """
    D_inf = []
    D_capacity = []
    epsilonBar = []
    delta = 1.e-100  # Add to make log finite

    P_n = GetPn(mu, epsilonList, Niter)

    Nboxes = [] # Number of non-zero P_n
    S0 = [] # Zero-dimensional entropy -sum(P_n log(P_n))

    for eps in epsilonList:
        Nboxes.append(sum(P_n[eps] > 0))
        S0.append(-sum(... * np.log(... + delta)))

    epsBar = []
    D_capacity = []
    D_inf = []
    for i in range(len(epsilonList) - 1):
        epsi = epsilonList[i]
        epsiP1 = epsilonList[i + 1]
        epsBar.append(np.sqrt(epsiP1 * epsi))
        D_capacity_estimate = -...
        D_capacity.append(D_capacity_estimate)
        D_inf_estimate = ...
        D_inf.append(D_inf_estimate)

    return epsBar, D_capacity, D_inf

muInfinity = 0.892486418
def PlotDimensionEstimates(mu=muInfinity, Niter=2**18,
                           epsilonList=2.0**np.arange(-4, -16, -1)):
```

```
    epsBar, DCapacity, DInformation = DimensionEstimates(mu,epsilonList
    plt.figure()
    plt.semilogx(epsBar, DCapacity, label='Capacity dim')
    plt.semilogx(epsBar, ..., label='Info dim')
    plt.title('Fractal dimension estimates, mu = '+ str (mu))
    plt.ylabel('Dimension')
    plt.xlabel('Bin size')
    plt.legend(loc=3)
```

In [ ]:
```
PlotDimensionEstimates(0.9)
PlotDimensionEstimates(0.8)
PlotDimensionEstimates(muInfinity)
```

Most 'real world' fractals have a whole spectrum of different characteristic spatial dimensions; they are *multifractal.*