Period doubling

(Sethna, "Entropy, Order Parameters, and Complexity", ex. 12.9)

In this exercise, we use renormalization-group and scaling methods to study the *onset of chaos*. There are several routes by which a dynamical system can start exhibiting chaotic motion; this exercise studies the *period-doubling cascade*, first extensively investigated by Feigenbaum.

Import packages

In [ ]:

```
%pylab inline
from scipy import *
from scipy.optimize import brentq
```

Chaos is often associated with dynamics which stretch and fold; when a batch of taffy is being pulled, the motion of a speck in the taffy depends sensitively on the initial conditions. A simple representation of this physics is provided by the map

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

restricted to the domain $(0,1)$. It takes $f(0) = f(1) = 0$, and $f(1/2) = \mu$. Thus, for $\mu = 1$ it precisely folds the unit interval in half, and stretches it to cover the original domain.

In [ ]:

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

The study of dynamical systems (*e.g.*, differential equations and maps like the logistic map $f(x)$ above often focuses on the behavior after long times, where the trajectory moves along the *attractor*. We can study the onset and behavior of chaos in our system by observing the evolution of the attractor as we change μ. For small enough μ, all points shrink to the origin; the origin is a stable fixed-point which attracts the entire interval $x \in (0,1)$. For larger μ, we first get a stable fixed-point inside the interval, and then *period doubling*.

(a) Iteration *Set $\mu = 0.2$; iterate f for some initial points x_0 of your choosing, and convince yourself that they all are attracted to zero. Plot f and the diagonal $y = x$ on the same plot. Are there any fixed-points other than $x = 0$? Repeat for $\mu = 0.3$, $\mu = 0.7$, and 0.8. What happens?*

Note: We write functions like Iterate abstractly, in terms of a function g(x,eta), so that we can also examine $f_{\sin}(x, B)$ where $\eta = B$ instead of $\eta = \mu$.

In [ ]:

```
def Iterate(x0, N, eta, g=f):
    """
    Iterate the function g(x,eta) N times, starting at x=x0.
    Return g(g(...(g(x))...)). Used to find a point on the attractor
    starting from some arbitrary point x0.

    Calling Iterate for the Feigenbaum map f^[1000] (x,0.9) at mu=0.9 woul
d look like
        Iterate(x, 1000, 0.9, f)

    We'll later be using Iterate to study the sine map
        fSin(x,B) = B sin(pi x)
    so passing in the function and arguments will be necessary for
    comparing the logistic map f to fSin.

    Inside Iterate you'll want to apply g(x0, eta).
    """
    for i in range(N):
        x0 = ...
    return x0

[print(mu, "->", Iterate(random.rand(4),100,mu,f)) for mu in [0.2,0.3,0.8,
0.9]];
```

In [ ]:

```
figure()
xs = arange(0,1,0.01)
plot(xs, xs)
for mu in (0.2, 0.3, 0.8, 0.9):
    plot(xs, f(...), label = "mu=%g" %mu)
legend(loc=2)
axes().set_aspect('equal')
```

On the same graph, plot f, the diagonal $y = x$, and the segments $\{x_0, x_0\}$, $\{x_0, f(x_0)\}$, $\{f(x_0), f(x_0)\}$, $\{f(x_0), f(f(x_0))\}$, ... (representing the convergence of the trajectory to the attractor). See how $\mu = 0.7$ and 0.8 differ. Try other values of μ.

In [ ]:

```
def IterateList(x,eta,Niter=10,Nskip=0,g=f):
    """
    Iterate the function g(x, eta) 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, g(x), g(g(x)), ... g(g(...(g(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
        pylab.hist(attractorXs, bins=500, normed=1)
        pylab.show()
    to see the density of points.
    """
    x = Iterate(x,Nskip,eta,g)
    xs = [x]
    for i in range(Niter-1):
        x = ...
        xs.append(x)
    return xs

def PlotIterate(mu,Niter=100,Nskip=0,x0=0.49):
    """
    Plots g, the diagonal y=x, and the boxes made of the segments
    [[x0,x0], [x0, g(x0)], [g(x0), g(x0)], [g(x0), g(g(x0))], ...

    Notice the xs and the ys are just the trajectory with each point
    repeated twice, where the xs drop the final point and the ys drop
    the initial point
    """
    figure()
    title('mu = '+ str (mu))
    xarray = arange(0.,1.,0.01)
    plot(xarray,xarray,'r-')   # Plot diagonal
    plot(xarray,...,'g-',linewidth=3)  # Plot function
    traj = IterateList(x0,mu,Niter,Nskip)
    doubletraj = array([traj,traj]).transpose().flatten()
    xs = doubletraj[...] # Drops last point
    ys = doubletraj[...]  # Drops first point
    plot(xs, ys, 'b-', linewidth=1, antialiased=True)
    axes().set_aspect('equal')

for mu in (0.7, 0.75, 0.8, 0.9):
    PlotIterate(mu)
```

By iterating the map many times, find a point a_0 on the attractor. As above, then plot the successive iterates of a_0 for $\mu = 0.7, 0.8, 0.88, 0.89, 0.9,$ and 1.0.

In [ ]:

```
for mu in (0.7, 0.75, 0.8, 0.9):
    PlotIterate(mu, Nskip=...)
```

You can see at higher μ that the system no longer settles into a stationary state at long times. The fixed-point where $f(x) = x$ exists for all $\mu > 1/4$, but for larger μ it is no longer *stable*. If x^* is a fixed-point (so $f(x^*) = x^*$) we can add a small perturbation $f(x^* + \epsilon) \approx f(x^*) + f'(x^*)\epsilon = x^* + f'(x^*)\epsilon$; the fixed-point is stable (perturbations die away) if $|f'(x^*)| < 1$. (In a continuous evolution, perturbations die away if the Jacobian of the derivative at the fixed-point has all negative eigenvalues. For mappings, perturbations die away if all eigenvalues of the Jacobian have magnitude less than one.)

In this particular case, once the fixed-point goes unstable the motion after many iterations becomes periodic, repeating itself after *two* iterations of the map---so $f(f(x))$ has two new fixed-points. This is called *period doubling*. Notice that by the chain rule $df(f(x))/dx = f'(x)f'(f(x))$, and indeed

$$\frac{df^{[N]}}{dx} = \frac{df(f(\ldots f(x)\ldots))}{dx} \qquad = f'(x)f'(f(x))\ldots f'(f(\ldots f(x)\ldots)),$$

so the stability of a period-N orbit is determined by the product of the derivatives of f at each point along the orbit.

(b) Analytics: *Find the 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 the period-two cycle is stable for $3/4 < \mu < (1 + \sqrt{6})/4$.*

(c) Bifurcation diagram: *Plot the attractor as a function of μ, for $0 < \mu < 1$.. (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 $\delta\mu < 0.01$.) Also on the bifurcation diagram, plot the line $x = 1/2$ where $f(x)$ reaches its maximum.*

In [ ]:

```
def BifurcationDiagram(etaMin=0., etaMax=1., deltaEta=0.001, g=f, x0=0.49,
 Nskip=100, Niter=128):
    figure(figsize=(50,20))
    xlim((etaMin,etaMax))
    etaArray = arange(etaMin, etaMax, deltaEta)
    etas = []
    trajs = []
    for eta in etaArray:
        etas.extend([eta]*Niter)
        trajs.extend(IterateList(...))
    scatter(etas, trajs, marker = '.', s=0.2)
    plot([0.,1.],[0.5,0.5],'g-')
    setp(axes().get_xticklabels(),fontsize=40)
    setp(axes().get_yticklabels(),fontsize=40)
    ylabel('x', fontsize=40)
```

In [ ]:

```
BifurcationDiagram()
```

In [ ]:

```
BifurcationDiagram(...[Zoom in; decrease deltaEta and increase Nskip until
 it looks nice])
```

Also plot the attractor for another one-humped map

$$f_{\sin}(x) = B\sin(\pi x),$$

for $0

In [ ]:

```
def fSin(x, B):
    return ...

BifurcationDiagram(g=fSin)
```

Notice the complex, structured, chaotic region for large μ. How do we get from a stable fixed-point $\mu < 3/4$ to chaos? The onset of chaos in this system occurs through a cascade of period doublings. There is the sequence of bifurcations as μ increases---the period-two cycle starting at $\mu_1 = 3/4$, followed by a period-four cycle starting at μ_2, period-eight at μ_3---a whole period-doubling cascade. The convergence appears geometrical, to a fixed-point μ_∞: \begin{equation} \mun \approx \mu\infty - A \delta^{-n}, \end{equation} so \begin{equation} \delta = \lim{n\to\infty} (\mu{n-1} - \mu{n-2})/(\mu_n - \mu{n-1}) \end{equation} and there is a similar geometrical self-similarity along the x axis, with a (negative) scale factor α relating each generation of the tree.

In the exercise 'Invariant Measures', we explained the boundaries in the chaotic region as images of $x = 1/2$. These special points are also convenient for studying period-doubling. Since $x = 1/2$ is the maximum in the curve, $f'(1/2) = 0$. If it were a fixed-point (as it is for $\mu = 1/2$), it would not only be stable, but unusually so: a shift by ϵ away from the fixed point converges after one step of the map to a distance $\epsilon f'(1/2) + \epsilon^2/2 f''(1/2) = O(\epsilon^2)$. We say that such a fixed-point is superstable. (The superstable points are the values of μ in the figure above which intersect the green line x=1/2.) If we have a period-N orbit that passes through $x = 1/2$, so that the Nth iterate $f^N(1/2) \equiv f(\ldots f(1/2) \ldots) = 1/2$, then the orbit is also superstable, since the derivative of the iterated map is the product of the derivatives along the orbit, and hence is also zero.

These superstable points happen roughly half-way between the period-doubling bifurcations, and are easier to locate, since we know that $x = 1/2$ is on the orbit. Let us use them to investigate the geometrical convergence and self-similarity of the period-doubling bifurcation diagram from part~(d). We will measure both the superstable values of μ and the size of the centermost 'leaf' in the bifurcation diagram (crossed by the line $x = 1/2$ where $f(x)$ takes its maximum). For this part and part~(h), you will need a routine that finds the roots $G(y) = 0$ for functions G of one variable y.

(d) The Feigenbaum numbers and universality: Numerically, find the values of μ_n^s at which the 2^n-cycle is superstable (the intersections of the attractor with the green line $x = 1/2$), for the first few values of n. (Hint: Define a function $G(\mu) = f_\mu^{[2^n]}(1/2) - 1/2$, and find the root as a function of μ. In searching for μ_{n+1}^s, you will want to search in a range $(\mu_n^s + \epsilon, \mu_n^s + (\mu_n^s - \mu_{n-1}^s)/A)$ where $A \sim 3$ works pretty well. Calculate μ_0^s and μ_1^s by hand.) Also, find the separation $1/2 - f^{[n-1]}(1/2, \mu_n^s)$, the opening between the two edges of the leaf crossed by the maximum of f (green line above).

In [ ]:

```
def FindSuperstable(g, Niter, etaMin, etaMax, xMax=0.5):
    """
    Finds superstable orbit g^[nIterated](xMax, eta) = xMax
    in range (etaMin, etaMax).
    Must be started with g-xMax of different sign at etaMin, etaMax.

    Notice that nIterated will usually be 2^n. (Sorry for using the
    variable n both places!)

    Uses optimize.brentq, defining a temporary function, G(eta)
```

```
    which is zero for the superstable value of eta.

    G iterates g nIterated times and subtracts xMax
    (basically Iterate - xMax, except with only one argument eta)
    """
    def G(eta):
        return Iterate(...)-xMax
    eta_root = brentq(G, etaMin, etaMax)
    return eta_root


def GetSuperstablePointsAndIntervals(g, eta0, eta1, nMax = 11, xMax=0.5):
    """
    Given the parameters for the first two superstable parameters eta_ss
[0]
    and eta_ss[1], finds the next nMax-1 of them up to eta_ss[nMax].
    Returns dictionary eta_ss

    Usage:
        Find the value of the parameter eta_ss[0] = eta0 for which the fix
ed
        point is xMax and g is superstable (g(xMax) = xMax), and the
        value eta_ss[1] = eta1 for which g(g(xMax)) = xMax, either
        analytically or using FindSuperstable by hand.
        mus = GetSuperstablePoints(f, 9, eta0, eta1)

    Searches for eta_ss[n] in the range (etaMin, etaMax),
    with etaMin = eta_ss[n-1] + epsilon and
    etaMax = eta_ss[n-1] + (eta_ss[n-1]-eta_ss[n-2])/A
    where A=3 works fine for the maps I've tried.
    (Asymptotically, A should be smaller than but comparable to delta.)
    """
    eps = 1.0e-10
    A = 3.
    eta_ss = {}
    eta_ss[0] = eta0
    eta_ss[1] = eta1
    leafSize = {}
    leafSize[1] = ...
    for n in arange(2, nMax+1):
        etaMin = ...
        etaMax = ...
        nIterated = 2**n
        eta_ss[n] = FindSuperstable(...)
        leafSize[n] = Iterate(xMax, 2**(...), eta_ss[n], g)-xMax
    return eta_ss, leafSize

mu0 = FindSuperstable(f,1,0.3,1.0,0.5)
mu1 = FindSuperstable(f,2,mu0 + 1.e-6,1.0,0.5)
mus, leafSizes = GetSuperstablePointsAndIntervals(f,mu0,mu1)
```

```
print(mus)
print(leafSizes)
```

Calculate the ratios $(\mu^s_{n-1} - \mu^s_{n-2})/(\mu^s_n - \mu^s_{n-1})$; do they appear to converge to the Feigenbaum number $\delta = 4.6692016091029909\ldots$? Estimate μ_∞ by using your last two values of μ^s_n, your last ratio estimate of δ, and the equations $\mu_n \approx \mu_\infty - A\delta^{-n}$ and $\delta = \lim_{n\to\infty}(\mu_{n-1} - \mu_{n-2})/(\mu_n - \mu_{n-1})$ above. In the superstable orbit with 2^n points, the nearest point to $x = 1/2$ is $f^{[2^{n-1}]}(1/2)$. (This is true because, at the previous superstable orbit, 2^{n-1} iterates returned us to the original point $x = 1/2$.) Calculate the ratios of the amplitudes $f^{[2^{n-1}]}(1/2) - 1/2$ at successive values of n; do they appear to converge to the universal value $\alpha = -2.50290787509589284\ldots$?

In [ ]:

```
nMax = 11;     # Note: Value of mu for n>9 not reliable

def ExponentEstimates(g, eta_ss, leafSizes, xMax=0.5, nMax=nMax):
    """
    Given superstable 2^n cycle values eta_ss[n], calculates
    delta[n] = (eta_{n-1}-eta_{n-2})/(eta_{n}-eta_{n-1})
    and alpha[n] = leafSizes[n-1]/leafSizes[n]

    Also extrapolates eta to etaInfinity using definition of delta and
    most reliable value for delta:
    delta = lim{n->infinity} (eta_{n-1}-eta_{n-2})/(eta_n - eta_{n-1})

    Returns delta and alpha dictionaries for n>=2, and etaInfinity
    """
    delta = {}
    alpha = {}
    for n in range(2, nMax+1):
        delta[n] = ...
        alpha[n] = ...
    etaInfinity = eta_ss[nMax] + ...
    return delta, alpha, etaInfinity

deltas, alphas, muInfinity = ExponentEstimates(f, mus, leafSizes, nMax=11)

deltas[nMax], alphas[nMax], muInfinity
```

Calculate the same ratios for the map $f_2(x) = B\sin(\pi x)$; do α and δ appear to be universal (independent of the mapping)?

In [ ]:

```
B0 = FindSuperstable(fSin,1,...)
B1 = FindSuperstable(fSin,2,B0+1.e-6,...)
Bs, leafSizesSin = GetSuperstablePointsAndIntervals(...)
deltaSin, alphaSin, BInfinity = ExponentEstimates(...)

deltaSin[nMax], alphaSin[nMax], BInfinity
```

The limits α and δ are independent of the map, so long as it folds (one hump) with a quadratic maximum. They are the same, also, for experimental systems with many degrees of freedom which undergo the period-doubling cascade. This self-similarity and universality suggests that we should look for a renormalization-group explanation.

(e) Coarse-graining in time. Plot $f(f(x))$ vs.\ x for $\mu = 0.8$, together with the line $y = x$. Notice that the period-two cycle of f becomes a pair of stable fixed-points for $f^{[2]}$.<\em> (We are coarse-graining in time---removing every other point in the time series, by studying $f(f(x))$ rather than f.) Compare the plot with that for $f(x)$ vs.\ x for $\mu = 0.5$. Notice that the region zoomed in around $x = 1/2$ for $f^{[2]} = f(f(x))$ looks quite a bit like the entire map f at the smaller value $\mu = 0.5$. Plot $f^{[4]}(x)$ at $\mu = 0.875$; notice again the small one-humped map near $x = 1/2$.

In [ ]:

```
x = arange(0.,1.,0.01)
plot(...))
plot(...)
title('mu='+str(0.8))
axes().set_aspect('equal')

figure()
plot(x,f(x,0.5))
plot(x,x)
title('mu='+str(0.5))
axes().set_aspect('equal')

figure()
plot(...)
plot(...)
title('mu='+str(0.875))
axes().set_aspect('equal')
```

The fact that the one-humped map reappears in smaller form just after the period-doubling bifurcation is the basic reason that succeeding bifurcations so often follow one another. The fact that many things are universal is due to the fact that the little one-humped maps have a shape which becomes independent of the original map after several period-doublings.

Let us define this renormalization-group transformation $\mathcal{T}$, taking function space into itself. Roughly speaking, $\mathcal{T}$ will take the small upside-down hump in $f(f(x))$, invert it, and stretch it to cover the interval from $(0,1)$. Notice in your graphs for part~(g) that the line $y = x$ crosses the plot $f(f(x))$ not only at the two points on the period-two attractor, but also (naturally) at the old fixed-point $x^*[f]$ for $f(x)$. This unstable fixed-point plays the role for $f^{[2]}$ that the origin played for f; our renormalization-group rescaling must map $(x^*[f], f(x^*)) = (x^*, x^*)$ to the origin. The corner of the window that maps to $(1,0)$ is conveniently located at $1 - x^*$, since our map happens to be symmetric about $x = 1/2$. (For asymmetric maps, we would need to locate this other corner $f(f(x_c)) = x^*$ numerically. As it happens, breaking this symmetry is irrelevant at the fixed-point.) For a general one-humped map $g(x)$ with fixed-point $x^*[g]$ the side of the window is thus of length $2(x^*[g] - 1/2)$. To invert and stretch, we must thus rescale by a factor $\alpha[g] = -1/(2(x^*[g] - 1/2))$. Our renormalization-group transformation is thus a mapping $\mathcal{T}[g]$ taking function space into itself, where
\begin{equation} T<u>g (x)</u> = \alpha[g] \left(g\left(g(x/\alpha[g]+x^[g])\right) - x^[g]\right). \end{equation}
(This is just rescaling x to squeeze into the window, applying g twice, shifting the corner of the window to the origin, and then rescaling by α to fill the original range $(0,1) \times (0,1)$.)

(f) Scaling and the renormalization group: Write routines that calculate $x^*[g]$ and $\alpha[g]$, and define the renormalization-group transformation $\mathcal{T}[g]$. Plot $\mathcal{T}[f]$, $\mathcal{T}[\mathcal{T}[f]]$,... and compare them. Are we approaching a fixed-point f^* in function space?

In [ ]:

```
def XStar(g, mu):
    """
    Finds fixed point of one-humped map g, which is assumed to be
    between xMax and 1.0.
    """
    def gXStar(x,mu): return ...
    return brentq(gXStar, 1/2, 1.0, args=(mu,))

def Alpha(g, mu):
    """
    Finds the (negative) scale factor alpha which inverts and rescales
    the small inverted region of g(g(x)) running from (1-x*) to x*.
    """
    gWindowMax = XStar(g, mu)
    gWindowMin = ...
    return 1.0/(gWindowMin-gWindowMax)

class T:
    """
    Creates a new function T[g] from g, implementing Feigenbaum's
    renormalization-group transformation of function space into itself.

    We define it as a class so that we can initialize alpha and xStar,
    which otherwise would need to be recalculated each time T[g] was
    evaluated at a point x.

    Usage:
        Tg = T(g, args)
        Tg(x) evaluates the function at x
    """
    def __init__(self, g, mu):
        """
        Stores g and args.
        Calculates and stores xStar and alpha.
        """
        self.mu = mu
        self.xStar = XStar(g, mu)
        self.alpha = Alpha(g, mu)
        self.g = g

    def __call__(self, x, mu):
        """
        Defines xShrunk to be x/alpha + x*
        Evaluates g2 = g(g(xShrunk))
        Returns expanded alpha*(g2-xStar)
        """
        xShrunk = .../self.... + self....
        g2 = self.g(self.g(...), ...)
        return ... * (g2 - ...)
```