The onset of chaos: Full renormalization-group calculation

(Sethna, "Entropy, Order Parameters, and Complexity", ex. 12.XXX)

In this exercise, we implement Feigenbaum's numerical scheme for finding high-precision values of the universal constants $\alpha = -2.5029078750958928222839028732 2\ \delta = 4.6692$ that quantify the scaling properties of the period-doubling route to chaos (Fig. 12.17, Exercise 'Period doubling'). This extends the lowest-order calculation of the companion Exercise 'The onset of chaos: Lowest order renormalization-group for period doubling'}.

Import packages

In [ ]:

```
%pylab inline
from scipy import *
from scipy.optimize import root
from scipy.linalg import eig

alphaFeigenbaum = -2.5029078750958928222839028732
18
deltaFeigenbaum = 4.6692016091029906718532038215
78
```

Our renormalization group operation (Exercises 'Period doubling and the renormalization group' and the companion Exercise) coarse-grains in time taking $g \to g \circ g$, and then rescales distance x by a factor of α. Centering our functions at $x = 0$, this leads to $T[g](x) = \alpha g\left(g(x/\alpha)\right)$.

We shall solve for the properties at the onset of chaos by analyzing our function-space renormalization-group by expanding our functions in a power series

$$g(x) \approx 1 + \sum_{n=1}^{N} G_n x^{2n}.$$

Notice that we only keep even powers of x; the fixed point is known to be symmetric about the maximum, and the unstable mode responsible for the exponent δ will also be symmetric.

In [ ]:

```
def g(G,x):
```



```
    """
    Returns 1 + G[0] x^2 + G[1] x^4 + ..., where
 G_n = G[n-1]
    We will sometimes call g with a whole array
 of x-values.
    """
    # enumerate(G) = [[0,G[0]], [1,G[1]], ...],
 conveniently giving n-1 and Gn in the formula.
    # enumerate(G,1) starts the numbering at one
    # sum(M) adds up all the entries of a matri
x. This is OK if x is a scalar, but if we send i
n a whole
    # array [x1,x2,...] we want an array of valu
es [g(x1),g(x2),...]. sum(M,axis=0) sums up the
 rows of the matrix.
    return 1.+sum([... for n,Gn in
enumerate(G,1)],axis=0)

def T(g,G,x,alpha=None):
    """
    Returns renormalization-group transform T[g]
(x).
    If alpha is not known, calculate it from g u
sing your result from (a) below.
    """
    if alpha is None:
        alpha = ...
    return ...

def Dg(G,x):
    """
    Returns g'(x)
    """
    return sum(...)
```

```
# Test your functions by plotting them. G = [-1.
5, 0, 0, ...] should give T[g] close to g for x<
1
x = arange(0,3,0.01)
plot(x,g([-1.5,0.],x))
plot(x,T(g,[-1.5,0],x))
```

First, we must approximate the fixed point $g^*(x)$ and the corresponding value of the universal constant α. At order N, we must solve for α and the N polynomial coefficients G_n^*. We can use the $N+1$ equations fixing the function at equally spaced points in the positive unit interval: \begin{equation} Tg^* (x_m) = g^*(x_m), ~~~~ x_m = m/N,~m = {0,\dots,N}. \end{equation} We can use the first of these equations to solve for α.

(a) *Show that the equation for $m = 0$ sets $\alpha = 1/g^*(1)$.*

We can use a root-finding routine to solve for G_n^*.

(b) *Implement the other N constraint equations above in a form appropriate for your method of finding roots of nonlinear equations, substituting your value for α from part (a). Check that your routine at $N = 1$ gives values for $\alpha \approx -2.5$ and $G_1^* \approx -1.5$.* (These should reproduce the values from the companion Exercise part (c).)

In [ ]:

```
def toZero(G):
    """Returns T[g](x) - g(x) for N points [1/N,
2/N,...,1], given N terms in its power series G
n"""
    N = len(G)
    x = linspace(...)
    return ...

# Check that your return gives a sensible value
 for the difference of Tg and g at x=1, for G1 =
 -1.5
print(toZero([-1.5]))

# Use root to find the best solution for N=1. Th
e values giving zero is returned as root(toZero,
[initial values]).x
G1 = root(...,[-1.5]).x

# What do we get for alpha[1]?
1/...
```

(c) *Use a root-finding routine to calculate α for $N = 1, \dots, 9$. Start the search at $G_1^* = -1.5$, $G_n^* = 0$ $(n > 1)$ to avoid landing at the wrong fixed point.* (If it is convenient for you to use high-precision arithmetic, continue to higher N.) *To how many decimal places can you reproduce the correct value for α at the beginning of this exercise?*

In [ ]:

```
# Fill dictionary with your values of alpha[N] f
or N = 1...9
# Also keep your values for the fixed point func
tion G[N] for use in calculating delta
alpha = {}
G = ...
Nmax = 9
for N in range(1,Nmax):
    G0 = zeros(N)
    G0[0]=-1.5
    G[N] = root(...
    alpha[N] = ...

# Print out your alphas
print(array([(N,...) for N in range(1,Nmax)]))

# Calculate how far they deviate from alphaFeige
nbaum
[(N,alphaFeigenbaum-...
```

Now we need to solve for the renormalization group flows $T[g]$, linearized about the fixed point $g(x) = g^*(x) + \epsilon\psi(x)$. Feigenbaum tells us that $T[g^* + \epsilon\psi] = T[g^*] + \epsilon\mathcal{L}[\psi]$, where $\mathcal{L}$ is the linear operator taking $\psi(x)$ to \begin{equation} \mathcal{L}\psi (x) = \alpha \psi(g^(x/\alpha)) + \alpha {g^}'(g(x/\alpha) \psi(x/\alpha). \end{equation}

(d) *Derive the equation above.*

We want to find eigenfunctions that satisfy $\mathcal{L}[\psi] = \lambda\psi$. Again, we can expand $\psi(x)$ in a polynomial \begin{equation} \psi(x) = \sum{n=0}^{N-1} \psi_n x^{2n} —- (\psi_0 \equiv 1). \end{equation} We then approximate the action of $\mathcal{L}$ on ψ by its action at N points $\tilde{x}_i$, that need not be the same as the N points x_m we used to find g^*. We shall use $\tilde{x}_i = (i-1)/(N-1)$, $i = 1, \dots, N$. (For $N = 1$, we use $\tilde{x}_1 = 0$.) This leads us to a linear system of N equations for the coefficients ψ_n, using the previous two equations. \begin{equation} \sum{n=0}^{N-1} \left[\alpha g(\tilde{x}i/\alpha)^{2 n} + \alpha g'(g(\tilde{x}_i/\alpha)) (\tilde{x}_i/\alpha)^{2n}\right]\psi_n = \lambda \sum{n=0}^{N-1} \tilde{x}i^{2 n} \psi_n \end{equation} These equations for the coefficients ψ_n of the eigenfunctions of $\mathcal{L}$ is in the form of a generalized eigenvalue problem \begin{equation} \sum_n L{in} \psin = \lambda X{in} \psi_n. \end{equation} The solution to the generalized eigenvalue problem can be found from the eigenvalues of $X^{-1}L$, but most eigenvalue routines provide a more efficient and accurate option for directly solving the generalized equation given L and X.

(e) *Write a routine that calculates the matrices L and X implicitly defined by the previous two equations. For $N = 1$ you should generate 1×1 matrices. For $N = 1$, what is your prediction for δ?* (These should reproduce the values from the companion Exercise part (d).)

In [ ]:

```
def X(N):
    """Returns X_{in} = xtilde_i**(2n)"""
    # Make sure your matrix hasn't transposed rows (i) and columns (n). Each row should have
    xtildes = linspace(0.,1.,N)
    return array([[... for n in range(N)] for xtilde in xtildes])

print(X(1))
print(X(3))

def Ln(xtildes,n,alpha,G):
    """Returns one column of L, given the array of xtilde values"""
    return alpha*g(...)**(...)+alpha*Dg(...)*(...)**(...)

# Test Ln on the one-element column for N=1: does it give a reasonable value for delta?
print('delta[1] should be the entry in ', Ln(array([0.]),0,alpha[1],G[1]))

def L(N):
    """Builds an array Lin from the columns Ln"""
    # Again, make sure your matrix has rows (i) and columns (n). You may need to use M.transpose.
    xtildes = ...
    return array([Ln(...) for n in range(N)]).transpose()

print(L(1))

print(L(3))

eig(L(3),X(3))
```