The onset of chaos: Full renormalization-group calculation

(Sethna, "Entropy, Order Parameters, and Complexity", ex. 12.30)

In this exercise, we implement Feigenbaum's numerical scheme for finding high-precision values of the universal constants

$$\alpha = -2.50290787509589282228390287322$$
$$\delta = 4.66920160910299067185320382158,$$

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 12.29 'The onset of chaos: Lowest order renormalization-group for period doubling'}.

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
from scipy.optimize import root
from scipy.linalg import eig

alphaFeigenbaum = -2.50290787509589282228390287321 8
deltaFeigenbaum = 4.66920160910299067185320382157 8
```

Our renormalization group operation (Exercises 'Period doubling and the renormalization group' and the companion Exercise 12.29) 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 $T[g](x) = \alpha g\,(g(x/\alpha))$.

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
    # enumerate(G,1) starts the numbering at one
    # sum(M) adds up all the entries of a matrix. This is OK if x is a
    # array [x1,x2,...] we want an array of values [g(x1),g(x2),...]. s
    return 1.+np.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 using your result from (
    """
    if alpha is None:
        alpha = ...
    return ...

def Dg(G,x):
    """
    Returns g'(x)
    """
    return np.sum(...,axis=0)

# Test your functions by plotting them. G = [-1.5, 0, 0, ...] should gi
x = np.arange(0,2,0.01)
plt.plot(x,g([-1.5,0.],x))
plt.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:

$$T[g^*](x_m) = g^*(x_m), \qquad x_m = m/N,\ m = \{0, \dots, N\}.$$

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 12.29 part (c).)

In [ ]:

```
def toZero(G):
    """Returns T[g](x) - g(x) for N points [1/N,2/N,...,1], given N ter
    N = len(G)
    x = np.linspace(...)
    return ...

# Check that your return gives a sensible value for the difference of T
print(toZero([-1.5]))

# Use root to find the best solution for N=1. The values giving zero is
G1 = root(...,[-1.5]).x

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

(c) *Use a root-finding routine to calculate* α *for* $N = 1, \ldots, 9$. *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] for N = 1...9
# Also keep your values for the fixed point function Gstar[N]
# for use in calculating delta
alpha = {}
Gstar = ...
Nmax = 15
for N in range(1,Nmax):
    G0 = np.zeros(N)
    G0[0]=-1.5
    Gstar[N] = root(...).x
    alpha[N] = ...

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

# Calculate how far they deviate from alphaFeigenbaum
[(N,alphaFeigenbaum-...) for ...]
```

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

$$\mathcal{L}[\psi](x) = \alpha\psi(g^*(x/\alpha)) + \alpha g^{*\prime}(g(x/\alpha))\psi(x/\alpha).$$

(d) *Derive the equation above.*

[Answer here]

We want to find eigenfunctions that satisfy $\mathcal{L}[\psi] = \lambda\psi$. Again, we can expand $\psi(x)$ in a polynomial

$$\psi(x) = \sum_{n=0}^{N-1} \psi_n x^{2n} \qquad (\psi_0 \equiv 1).$$

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 $\tilde{x}_i = (i-1)/(N-1)$, $i = 1, \ldots, N$. (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.

$$\sum_{n=0}^{N-1} \left[\alpha g(\tilde{x}_i/\alpha)^{2n} + \alpha g'(g(\tilde{x}_i/\alpha))(\tilde{x}_i/\alpha)^{2n}\right] \psi_n = \lambda \sum_{n=0}^{N-1} \tilde{x}_i^{2n} \psi_n$$

These equations for the coefficients ψ_n of the eigenfunctions of $\mathcal{L}$ is in the form of a *generalized eigenvalue problem*

$$\sum_n L_{in}\psi_n = \sum_n \lambda X_{in}\psi_n.$$

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 12.29 part

In [ ]:
```
def X(N):
    """Returns X_{in} = xtilde_i**(2n)"""
    # Make sure your matrix hasn't transposed rows (i) and columns (n).
    xtildes = np.linspace(0.,1.,N)
    return np.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
print('delta[1] should be the entry in ', Ln(np.array([0.]),0,alpha[1],

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

print(L(1))

print(L(3))

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

(f) *Solve the generalized eigenvalue problem for L and X for $N = 1, \ldots, 9$. To how many decimal places can you reproduce the correct value for δ at the beginning of this exercise?*

In [ ]:

```
# Fill dictionary with your values of alpha[N] for N = 1...9
delta = {}
Nmax = 15
for N in range(1,Nmax):
    eigvals, eigvecs = ...
    delta[N] = np.real(eigvals[0])

# Print out your deltas
print(np.array([(...) for N in ...]))

# Calculate how far they deviate from deltaFeigenbaum
[(N,...) for N in range(1,Nmax)]
```