Uppsala University

Modelling Complex Systems

1MA256

Lab1 report

Author:
Kristensen.Samuel
Place of publication: Uppsala

April 14, 2024

1 Task 1: Bifurcation diagram

We modify the code from [1] to compute and draw the bifurcation diagram for the new map taking the sine map.

$$f_r(x) = rsin(\pi x) \tag{1}$$

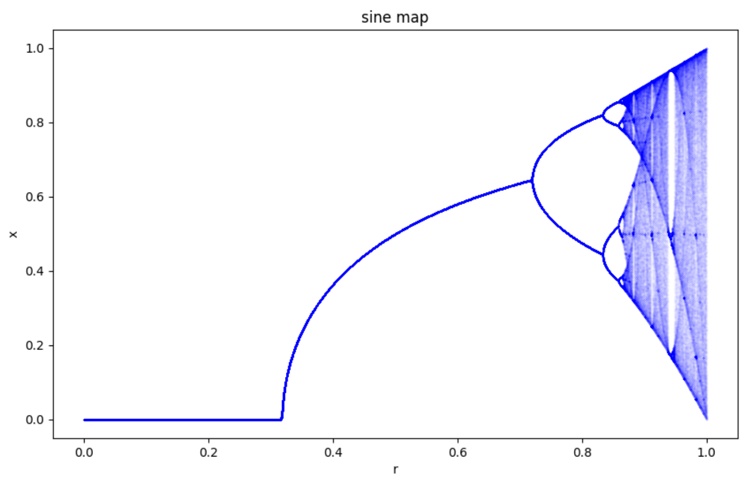


Figure 1: Bifurcation diagram for sine map

Figure 1 shows the bifurcation map for the sine function 1, when $x \in [0, 1]$. The code to draw the map is in section 5.1.

2 Task 2: Feigenbaum constants

We shall now calculate the Feigenbaum constants α and δ using the direct method and compare them to those for the logistic map.

We do this by measuring the distances between bifurcation parameters at which the 2^n-th orbit is superstable. We know that an orbit under a map q_μ is superstable if it contains point 0.5 and thus $q_\mu^{2^n}(0.5) = 0.5$. We are looking for μ so let it be a varaible and we write q_μ as $p(\mu, x)$:

$$p(\mu, x) = \mu sin(\pi x) \tag{2}$$

And the derivative of p with respect to μ and x is

$$\frac{dp(\mu, x)}{d\mu} = sin(\pi x), \qquad \frac{dp(\mu, x)}{dx} = \mu\pi cos(\pi x) \tag{3}$$

we then use the chain rule to differentiate

$$p(\mu, p(\mu, p(\mu, ..., p(\mu, 0.5)...))) - 0.5 = 0 \tag{4}$$

We then use Newtons method to find μ where equation 4 is true.

To get an appropriate starting value of μ we calculate μ_0 and μ_1 by hand:

$$p(\mu, 0.5) - 0.5 = 0 \iff \mu sin(0.5\pi) - 0.5 = 0 \iff \mu = \frac{0.5}{sin(0.5\pi)} = 0.5 = \mu_0 \tag{5}$$

$$p(\mu, p(\mu, 0.5)) - 0.5 = 0 \iff \mu sin(\pi \mu sin(0.5\pi)) - 0.5 = 0 \implies \mu = 0.777... = \mu_1 \tag{6}$$

Where the last equation is solved with wolfram alpha.

We can now solve with the newton method using

$$\mu_{n+1} = \mu_n + \frac{\mu_n - \mu_{n-1}}{\delta} \tag{7}$$

We also compute feigenbaum's α with

$$\alpha = \lim_{i \to \infty} \frac{b'_{i+1}(a_{i+1})}{b'_i(a_i)} \tag{8}$$

The code for this is in section 5.2 and the results when comparing the logistic map and the sin map were

	feigenbaum for logistic:			feigenbaum for sin:		
i	a_i	delta_i	alpha_i	a_i	delta_i	alpha_i
2	3.49856170	4.70898700	-2.44435563	0.84638217	4.04326179	-2.37284009
3	3.55464086	4.68073493	-2.48672317	0.86145035	4.55809366	-2.47530540
4	3.56666738	4.66295977	-2.49979063	0.86469418	4.64518181	-2.49700105
5	3.56924353	4.66840392	-2.50219678	0.86538967	4.66407540	-2.50164455
6	3.56979529	4.66895529	-2.50276080	0.86553866	4.66811188	-2.50263706
7	3.56991346	4.66916320	-2.50287554	0.86557057	4.66899156	-2.50284917
8	3.56993877	4.66918869	-2.50289986	0.86557741	4.66915149	-2.50289062
9	3.56994419	4.66916007	-2.50289961	0.86557887	4.66903888	-2.50287627
10	3.56994535	4.66906797	-2.50287740	0.86557918	4.66867530	-2.50277816
11	3.56994560	4.66876484	-2.50277309	0.86557925	4.66747582	-2.50233024
12	3.56994566	4.66775237	-2.50230134	0.86557926	4.66346506	-2.50029866
13	3.56994567	4.66433701	-2.50014625	0.86557927	4.64984999	-2.49101215

And we see that for both the logistic map and the sin map the feigenbaum delta value will converge to almost the same and close to the theoretical value of 4.669. The alpha value for both will also converge to almost the same and is similar to the theoretical although negative, not quite sure why.

3 Task 3: Lyapunov exponents

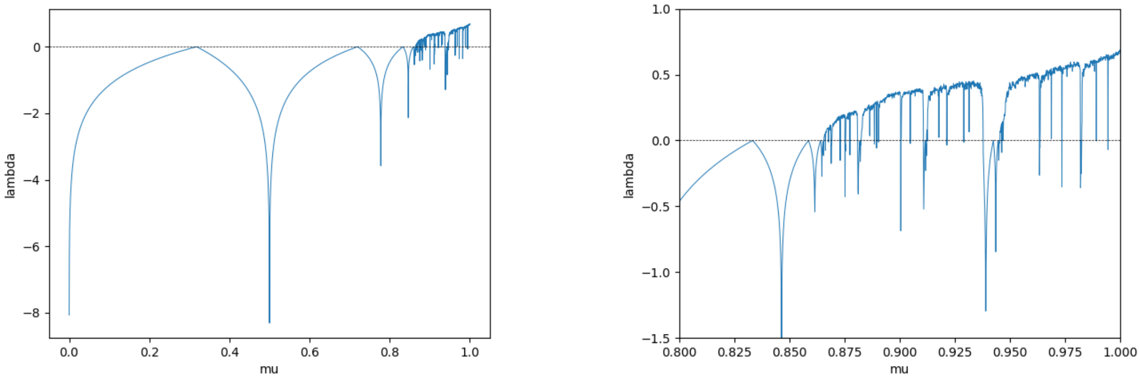
We now calculate the lyapunov exponent λ to characterise the chaos. It is calculated using the formula [2]:

$$\lambda = \lim_{n \to \infty} \frac{1}{n} \sum_{i=0}^{n-1} ln|f'(x_i)|. \tag{9}$$

where

$$f'(x) = \frac{d}{dx} \mu sin(\pi x) = \mu \pi cos(\pi x) \tag{10}$$

We range μ from 0 to 1, we initialize x to a random number and let it step 1000 steps before we start the process to ignore the transient effect. The code for the experiments is in section 5.3.



(a) Lyapunov exponents

(b) Lyapunov exponents closeup

Figure 2: Lyapunov exponents for $0 \leq \mu \leq 1$ with closeup to the right

In figure 2 we see the lyapunov exponents for the sine map when $0 \leq \mu \leq 1$ and also the closeup for the chaotic area. The negative values corresponds to when the system is stable and the positivive values corresponds to when the system is chaotic. We can especially see that the two first "deep" points (very negative) in the left figure is $\mu \approx 0.5$ and $\mu \approx 0.78$ which is also the first two points that we knew were superstable from section 2, i.e. μ_0 and μ_1. If we compare the figure with the bifurication diagram 1 we can see that where $\lambda > 0$ is where the bifurication diagram starts getting chaotic as-well. Also the periodic windows in the bifurcation diagram correspond to regions where the system is stable, which is why the Lyapunov exponent is negative there.

4 Task 4: Identifying a period 3 orbit

We want to fin a period 3 orbit, to start we need to find the two preimages of

$$\left(\frac{1}{\mu}, \frac{1 - \sqrt{1 - \frac{4}{\mu}}}{2} \right) \tag{11}$$

I.e. the two images which are mapped onto that interval by q_μ.

Note: I didnt really understand this task but i tried my best in understanding (i was sick the lecture where we showed that it "covers" itself after two iterates so i didnt have any notes on that and couldnt find any good on it). The square root demands that $\mu \geq 4$?

To find the two images that maps onto the interval in equation 11 we can iterate through x $\in$ [0,1] and see which values of x that maps onto the interval. We use $\mu = 4$ and from the code in section 5.4 we get that.

$$\text{Preimage 1} = (0.067, 0.1464) \qquad \text{Preimage 2} = (0.8536, 0.933) \tag{12}$$

5 Codes

5.1 Code for Task 1

```
import numpy as np
import matplotlib.pyplot as plt
import math
from collections import Counter

def sine_map(r, x):
    return r * np.sin(np.pi * x)

interval = (0.5, 4) # start, end
accuracy = 0.0001
reps = 600 # number of repetitions
numtoplot = 200
lims = np.zeros(reps) # list of 600 zeros

fig, biax = plt.subplots()
fig.set_size_inches(10, 6)

lims[0] = np.random.rand()
for r in np.arange(interval[0], interval[1], accuracy):
    for i in range(reps-1):
        lims[i+1] = sine_map(r, lims[i])

    biax.plot([r]*numtoplot, lims[reps-numtoplot:], 'b.', markersize=0.02)

biax.set(xlabel='r', ylabel='x', title='sine map')
plt.show()
```

5.2 Code for Task 2

```
import numpy as np
from scipy.optimize import newton

def logistic_map(mu, x):
    return mu * x * (1 - x)

def dlogistic_map(mu, x):
    return mu * (1 - 2 * x)

def sine_map(mu,x):
    return mu*np.sin(np.pi*x)

def sine_map_deriv(mu,x):
    return mu*np.pi*np.cos(np.pi*x)


def f_logistic(n, mu):
```

```
    logistic_val = np.zeros(2**n)
    logistic_val[0] = logistic_map(mu, 0.5)
    dlogistic_val = 0.25
    for i in range(1, 2**n):
        logistic_val[i] = logistic_map(mu, logistic_val[i-1])
        devpart = dlogistic_map(mu, logistic_val[i-1])
        dlogistic_val *= devpart
    return logistic_val[-1] - 0.5, dlogistic_val

def f_sin(n,mu):
    sine_val = np.zeros(2**n)
    sine_val[0] = sine_map(mu,0.5)
    sine_val_deriv = np.sin(np.pi*0.5)
    for i in range(1,2**n):
        sine_val[i] = sine_map(mu, sine_val[i-1])
        sine_devpart = sine_map_deriv(mu,sine_val[i-1])
        sine_val_deriv *= sine_devpart
    return sine_val[-1] -0.5, sine_val_deriv


def run_feigenbaum():
    iterations = 14
    deltareal = 4.6692016

    mu_values = np.zeros(iterations)
    mu_values[0] = 2
    mu_values[1] = 3.23607

    mu_sin_values = np.zeros(iterations)
    mu_sin_values[0] = 0.5
    mu_sin_values[1] = 0.7777

    print("feigenbaum for logistic:")
    print(" i       a_i        delta_i       alpha_i")
    for n in range(2, iterations):
        mu0 = mu_values[n-1] + (mu_values[n-1] - mu_values[n-2]) / deltareal
        mu_values[n] = newton(lambda mu: f_logistic(n, mu)[0], mu0,
            fprime=lambda mu: f_logistic(n, mu)[1],tol=1e-12 ,maxiter=10000)
        b0 = f_logistic(n-1,mu_values[n-1])[1]
        b1 = f_logistic(n,mu_values[n])[1]
        alpha = b1/b0
        d = (mu_values[n-1]-mu_values[n-2])/(mu_values[n]-mu_values[n-1])
        print("%2d   %1.8f   %1.8f   %1.8f" % (n, mu_values[n], d, alpha))

    print()
    print("feigenbaum for sin:")
    print(" i       a_i        delta_i       alpha_i")
    for n in range(2,iterations):
        mu0 = mu_sin_values[n-1] + (mu_sin_values[n-1] - mu_sin_values[n-2]) /
            deltareal
```

```
        mu_sin_values[n] = newton(lambda mu: f_sin(n, mu)[0], mu0, fprime=lambda
            mu: f_sin(n, mu)[1],tol=1e-12 ,maxiter=10000)
        d =
            (mu_sin_values[n-1]-mu_sin_values[n-2])/(mu_sin_values[n]-mu_sin_values[n-1])
        b0 = f_sin(n-1,mu_sin_values[n-1])[1]
        b1 = f_sin(n,mu_sin_values[n])[1]
        alpha = b1/b0
        print("%2d   %1.8f   %1.8f   %1.8f" % (n, mu_sin_values[n], d, alpha))

run_feigenbaum()
```

5.3 Code for Task 3

```
import numpy as np
import matplotlib.pyplot as plt


def sine_map(mu, x):
    return mu*np.sin(np.pi*x)

def sine_map_deriv(mu, x):
    return mu*np.pi*np.cos(np.pi*x)

def lyapunov():
    mu_values = np.linspace(0, 1, 10000)
    lambda_values = []

    for mu in mu_values:
        x = np.random.random()
        for _ in range(1000): # Discard the first 1000 iterations
            x = sine_map(mu, x)
        sum = 0
        for _ in range(1000): # Next 10000 iterations
            x = sine_map(mu, x)
            sum += np.log(abs(sine_map_deriv(mu, x)))
        lambda_values.append(sum / 1000)


    plt.plot(mu_values, lambda_values, linewidth=0.7)
    plt.axhline(y=0, color='black', linestyle='--',linewidth=0.5)
    plt.xlabel('mu')
    plt.ylabel('lambda')
    plt.show()

lyapunov()
```

5.4 Code for Task 4

```
import math
```

```
def q_mu(x, mu):
    return mu*x*(1-x)

def find_preimages(interval, mu, precision=0.0001):
    preimages = []
    for x in range(0, 10000):
        x_val = x * precision
        if interval[0] <= q_mu(x_val, mu) <= interval[1]:
            preimages.append(x_val)
    return preimages

def iterate_q(x, mu, n):
    for _ in range(n):
        x = q_mu(x, mu)
    return x

def find_period_3_orbit(mu):
    interval = (1/mu, (1 - math.sqrt(1 - 4/mu)) / 2)
    preimages = find_preimages(interval, mu)

    half = len(preimages)//2
    preimage1 = [preimages[0], preimages[half-1]]
    preimage2 = [preimages[half], preimages[-1]]
    print("Preimage 1 for the interval: ",preimage1, "Preimage 2 for the
        interval:", preimage2)
    return

mu_value = 4
period_3_orbit = find_period_3_orbit(mu_value)
```

References

[1] Wikipedia. Logistic map. https://en.wikipedia.org/wiki/Logistic_map.

[2] L2 Chapter 4. Introduction to chaotic dynamics. https://people.smp.uq.edu.au/MatthewDavis/phys2100/PHYS2100_HM_week3.pdf.