Homework 5,
Information theoretic analysis of the logistic map

Fredrik Sitje
950314-8232

March 2024

Definition of the logistic map

$$x(t+1) = rx(1-x) \tag{1}$$

where we'll use $r = 4$ for which x always lies in the range $[0, 1]$.

a)

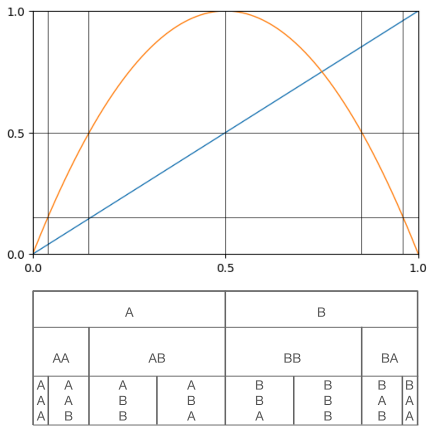

Figure 1: In this figure a logistic map normalised to 1 is shown in orange. In a dashed blue line the there is an auxiliary line for analysis on the behaviour of the partition. In a black vertical line there is a $x = 0.5$ indicating where the logistic map has its maximum.

We'll now consider $A \in [0.0, 0.5]$ and $B \in \,]0.5, 1.0]$ in figure 1 to represent a partition to analyse. The homework suggests that this partition is *generating*. So let's quickly remind ourselves what this implies.
Generating partition have the property that for a sequence $x_1x_2x_3 \ldots x_m$ where $x_j \in [A, B]$ the corresponding sets decrease in size,

$$\text{diam}(x_1x_2x_3 \ldots x_m) \to 0, \;\text{ as } m \to \infty. \tag{2}$$

As one can see from figure 1 the sub sequences always split to new unique sub sequences for each step. There for the diameter shrinks like equation (2) demands for a generating partition.

b)

Applying the logistic map for 100'000 iteration steps with initial position $x_0 = 0.6$ and $r = 4$ produces the following probabilities:
Probability of A: 0.50152
Probability of B: 0.49849

c)

The ergodic invariant measure for given r is given by

$$\mu(r) = \frac{1}{\pi\sqrt{x(1-x)}}. \tag{3}$$

Using the formula for the Lyapunov exponent

$$\lambda = \lim_{t\to\infty} \frac{1}{t} \sum_{k=0}^{t-1} \ln|f'(x(k))| = \int \mathrm{d}x\, \mu(x) \ln|f'(x)| \tag{4}$$

and applying it to equation (3) one calculates the Lyapunov exponent to be

$$\lambda = \int_0^1 \frac{\ln|4((1-x)-x)|}{\pi\sqrt{x(1-x)}} dx = \ln(2). \tag{5}$$

For code see Appendix `Mathematica`-Code.

d)

Calculating the probabilities for the new partition on the time series data gives the following probabilities:
Probability of new A: 0.5
Probability of new B: 0.5.
Doing exactly the same with the old partitions gives the probabilities:
Probability of old A: 0.36373
Probability of old B: 0.63627
The probabilities of the old partition are skewed and do not match up with the simulated data, therefore the underlying dynamics must have changed.

For code see Appendix `Python`-Code.

Appendix `Mathematica`-Code.

```
r = 4;
logisticMap[x_] := r * x(1 - x);
mu[x_]:=1/Pi * Sqrt[x(1 - x)];

logisticMapPrime = D[logisticMap[x], x];

Integrate[mu[x] * Log[Abs[logisticMapPrime]], {x, 0, 1}]
```

Appendix Python-Code

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

def f(x, r):
    return r * x * (1 - x)

# Define the range of x values
x = np.linspace(0, 1, 100)

# Define the value of r
r = 2

# Calculate the corresponding y values
y = f(x, r)

# Scale the y values
y_scaled = y / np.max(y)

# Create a figure and two subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(6, 8))

# Plot the first function with vertical lines
ax1.plot(x, x, '-', lw=1)
ax1.plot(x, y_scaled, '-', lw=1)
ax1.axvline(0.5, color='black', lw=0.5)
ax1.axvline(0.039, color='black', lw=0.5)
ax1.axvline(0.145, color='black', lw=0.5)
ax1.axvline(0.853, color='black', lw=0.5)
ax1.axvline(0.959, color='black', lw=0.5)
ax1.axhline(0.5, color='black', lw=0.5)
ax1.axhline(0.152, color='black', lw=0.5)
ax1.grid(False)
ax1.set_ylim(0, 1)
ax1.set_xlim(0, 1)
ax1.set_xticks([0, 0.5, 1])
ax1.set_yticks([0, 0.5, 1])

# Plot the second function without boxing lines
ax2.axhline(0.8, color='black', lw=0.5)
ax2.axhline(0.6, color='black', lw=0.5)
ax2.axhline(0.4, color='black', lw=0.5)
ax2.grid(False)
ax2.set_ylim(0, 1)
ax2.set_xlim(0, 1)

# Remove ticks and tick labels from the second plot
ax2.set_xticks([])
ax2.set_yticks([])
ax2.set_xticklabels([])
ax2.set_yticklabels([])

# Remove spines from the second plot
ax2.spines['top'].set_visible(False)
ax2.spines['right'].set_visible(False)
ax2.spines['bottom'].set_visible(False)
ax2.spines['left'].set_visible(False)


# Adjust spacing between subplots
plt.subplots_adjust(hspace=0.1)

plt.show()


# %%
# Define the function f(x, r)
def f(x, r):
    return r * x * (1 - x)

# Define the parameters
steps = 10**5
x_0 = 0.6
A = 1
```

```
B = 0
r = 4  # You need to define the value of r

# Run the simulation
for i in range(steps):
    x_0 = f(x_0, r)
    if x_0 <= 0.5:
        A += 1
    else:
        B += 1

# Calculate probabilities
probA = A / steps
probB = B / steps

# Print probabilities
print(f'Probability of A: {probA}')
print(f'Probability of B: {probB}')


# %%
# read in the csv file pop_series.csv

oldA = [0.0,0.5]
oldB = [0.5,1.0]
newA = [0.324,0.6004]
newB = [0.7884864, 0.9]

counterOldA = 0
counterOldB = 0
counterNewA = 0
counterNewB = 0

# open the file and read content line by line
with open('pop_series.csv', 'r') as file:
    lines = file.readlines()
    for value in lines:
        if newA[0] <= float(value) <= newA[1]:
            counterNewA += 1
        if newB[0] < float(value) <= newB[1]:
            counterNewB += 1
        if oldA[0] <= float(value) <= oldA[1]:
            counterOldA += 1
        if oldB[0] < float(value) <= oldB[1]:
            counterOldB += 1


oldTotal = counterOldA + counterOldB
#print(counterOldA, counterOldB, oldTotal)
probOldA = counterOldA / oldTotal
probOldB = counterOldB / oldTotal

newTotal = counterNewA + counterNewB
#print(counterNewA, counterNewB, newTotal)
probNewA = counterNewA / newTotal
probNewB = counterNewB / newTotal

print(f'Probability of old A: {probOldA}')
print(f'Probability of old B: {probOldB}')
print('')
print(f'Probability of new A: {probNewA}')
print(f'Probability of new B: {probNewB}')
```