Logistic Map

The logistic equation is a recurrence relation given by $x_{n+1} = rx_n(1 - x_n)$. One of its main areas of application is in modelling population growths. Usually, we take $r > 0$ and $x \in [0, 1]$.

```
"""
logistic_map function is repeatedly applied starting with initial value
known as x_0.

xn is the value of logistic_map from last call.
r is the growth rate parameter.
"""
def logistic_map(xn, r):
    return r*xn*(1-xn)
```

Boundedness of the logistic map :

The logistic map always converges when $0 < r < 4$. We can analyse when the map gets unbounded (we assume $r \geq 0$ for simplicity). For $0 < r < 1$, it always reaches zero (as seen from the next problem) , hence we need not consider this case.

It is clear that once x_n becomes negative, it keeps becoming more and more negative, as both r and $(1 - x_n)$ are greater than 1. So, we need to find the values of r for which x_n may become negative.

Let's say in nth iteration x_n becomes negative for the first time. Then x_{n-1} has to be greater than 1. This is because, only the $(1 - x_{n-1})$ term can flip the sign of x_n. So we have the question :

When is $rx(1 - x) > 1$?

Solving for $rx(1 - x) = 1$, we get that :

$$x \in \left(\frac{1 - \sqrt{1 - \frac{4}{r}}}{2}, \frac{1 + \sqrt{1 - \frac{4}{r}}}{2} \right)$$

For x to be real, we require $r > 4$. Thus, for $r \in [0, 4]$, the relation is always bounded. It has been found that for $r > 4$, the recurrence is almost always unbounded irrespective of initial x_0.

We run a python program for $r = 3.4$ and $r = 4.1$, and $x_0 = 0.2$ to illustrate this.

```
from tabulate import tabulate

r = 3.4
x_0 = 0.2
output = []
for i in range(1000):
```

```
  x_0 = logistic_map(x_0, r)
output.append([3.4, 1000, x_0])

r = 4.1
x_0 = 0.2
for i in range(20):
  x_0 = logistic_map(x_0, r)

output.append([4.1, 20, x_0])
print((tabulate(output, headers = ['r', 'iterations', 'final value'])))
```

```
  r    iterations    final value
---  ------------  -------------
3.4          1000       0.842154
4.1            20    -inf
```

Observation :

As expected, for $r = 3.4$, it is indeed bounded even after quite a few iterations, and for $r = 4.1$, it shoots to $-\infty$ even for very few iterations.

Logistic map for $0 < r < 1$:

The value of x eventually goes to 0 after a large number of iterations, when the $r \in (0, 1)$. This happens irrespective of the initial x value. The following is a python program for $x_0 = 0.2$ and $r = 0.5$. We display the value of x after 1, 10, 50, 100, 500 and 2000 iterations. It is clear that x keeps decreasing till it reaches 0.

```
x_0 = 0.2
output = []
r = 0.5
output.append([0, x_0])
for i in range (1, 2001): #we carry out the iteration 2000 times.
  x_0 = logistic_map(x_0, r)
  if i in [1, 10, 50, 100, 500, 2000]: # we are displaying the value of x after the
    output.append([i, x_0])

print(tabulate(output, headers = ['iterations', 'value']))
```

```
  iterations        value
------------  -----------
           0  0.2
           1  0.08
          10  0.000133701
          50  1.21568e-16
         100  1.07974e-31
         500  4.1814e-152
        2000  0
```

Logistic map for $1 < r < 3$:

Here, x reaches a certain value several iterations. After this, even if we iterate the map, x value remains the same. For $1 < r < 3$, this constant value is $\frac{r-1}{r}$. We run a python program for $r = 1.5$ and different initial values x_0. We can see that the value coverges to $\frac{1}{3}$, even for few iterations

```
r = 1.5
# The elements in this list are the various initial values taken
x_0_values = [0.00001, 0.001, 0.1, 0.2, 0.5 , 0.7]

# list of lists for tabular output
output = []
for x_0 in x_0_values:
  val = x_0
  for i in range(5):
    val = logistic_map(val, r)
  output.append([x_0, 5, val])
  for i in range(25):
    val = logistic_map(val, r)
  output.append([x_0, 30, val])
  for i in range(20): # we show what happens to x after 5, 30 and 50 iterations
    val = logistic_map(val,r)
  output.append([x_0, 50, val])
print(tabulate(output, headers = ['initial x', 'number of iterations', 'final value
```

initial x	number of iterations	final value
1e-05	5	7.59275e-05
1e-05	30	0.315156
1e-05	50	0.333333
0.001	5	0.00749458
0.001	30	0.333326
0.001	50	0.333333
0.1	5	0.28469
0.1	30	0.333333
0.1	50	0.333333
0.2	5	0.32303
0.2	30	0.333333
0.2	50	0.333333
0.5	5	0.335405
0.5	30	0.333333
0.5	50	0.333333
0.7	5	0.332061
0.7	30	0.333333
0.7	50	0.333333

Logistic map for $r > 3$:

The following is a python program for different initial x_0, with $r = 3.2$. We iterate the logistic_map 1000 times and display the last five iterations for each x_0. We can see that x does not reach a constant value. It alternates between two values. Interestingly, these two values between which x alternates are the same, irrespective of the initial x_0 (about 0.513 and 0.799). Thus, we can say that these final **equillibrium values** depend only on r, and not on x_0.

However, iteration numbers corresponding to 0.513 and to 0.799 do depend on the initial x_0 (for example, every even iteration corresponds to 0.513 to $x_0 = 0.01$, but every even iteration corresponds to 0.799 for $x_0 = 0.2$).

It has been found that these two values between which x alternates are given by : $x_{final} = \frac{1}{2r} \cdot (r + 1 \pm \sqrt{(r-3)(r+1)})$. Putting r = 3.2, we do get the two values of x_{final} to be 0.799 and 0.513. This relation holds only when $r \in \left(3, 1+\sqrt{6}\right)$.

```
x_0_values = [0.00001,0.001, 0.2, 0.7]
r = 3.2
output = []
for x_0 in x_0_values:
  a = x_0
  for j in range (1000):
    x_0 = logistic_map(x_0, r)
    if j >= 995:
      output.append([a, j+1, x_0])
print((tabulate(output, headers = ['initial x', 'iterations', 'final value'])))
```

```
  initial x    iterations    final value
-----------  ------------  -------------
      1e-05           996       0.513045
      1e-05           997       0.799455
      1e-05           998       0.513045
      1e-05           999       0.799455
      1e-05          1000       0.513045
      0.001           996       0.513045
      0.001           997       0.799455
      0.001           998       0.513045
      0.001           999       0.799455
      0.001          1000       0.513045
      0.2             996       0.799455
      0.2             997       0.513045
      0.2             998       0.799455
      0.2             999       0.513045
      0.2            1000       0.799455
      0.7             996       0.799455
      0.7             997       0.513045
      0.7             998       0.799455
      0.7             999       0.513045
      0.7            1000       0.799455
```

References

- Wikipedia Logistic Map
- Wolfram Logistic Map
- Visualizing Chaos and Randomness