

PROJECT REPORT-I.1 CALCULUS(BTECH IT & MI)



CLUSTER INNOVATION CENTRE

TEAM MEMBERS-

Saransh Chopra, Raunak Singh, Onkar and Antas.

Acknowledgement

We would like to express our special thanks of gratitude to our mentor Dr Sonam Tanwar for guidance and supervision as well as providing necessary information regarding the project.

It would be a great pleasure for us to present our 1st sem project on calculus I.1.

**CLUSTER INNOVATION CENTRE
BTECH(IT&MI)
DELHI UNIVERSITY**

Certificate of completion

This certifies that the team members Saransh, Raunak, Onkar and Antas have completed the project under my guidance as per the necessities and requirements for 1st sem project in course BTech Information Technology and Mathematical Innovations, Cluster Innovation centre, University of Delhi.

Dr Sonam Tanwar
Department of Mathematics
Cluster Innovation Centre
Delhi University

CHAOS AND FRACTALS

CONNECTING CHAOS WITH FRACTALS



Introduction

We have attempted to discuss the various domains in which mathematics, and especially calculus finds itself in. We aim to showcase existing methods and systems and depict their usage in real world applications. Particularly *Chaos based Image Encryption*. We try to define the various systems we use and give a brief background regarding the same. We use the logistic growth equation to encrypt image pixels, the details of which are discussed subsequently.

Logistic map

The logistic map is widely used to show the properties of chaotic dynamics. A rough description of chaos is that chaotic systems exhibit a great sensitivity to initial conditions—a property of the logistic map for most values of r between about 3.57 and 4 (as noted above). A common source of such sensitivity to initial conditions is that the map represents a repeated folding and stretching of the space on which it is defined.

Chaos

Chaotic systems, in the field of mathematics, are generally systems defined by equations and exclusive parameters which display an unfathomable degree of randomness when plotted. They are extremely sensitive to initial conditions and an extremely minor change leads to a completely, seemingly new, plot. There are various examples of chaotic systems some of which are The Lorenz system, the Rossler system, the Van der pol model and many more.

Fractals

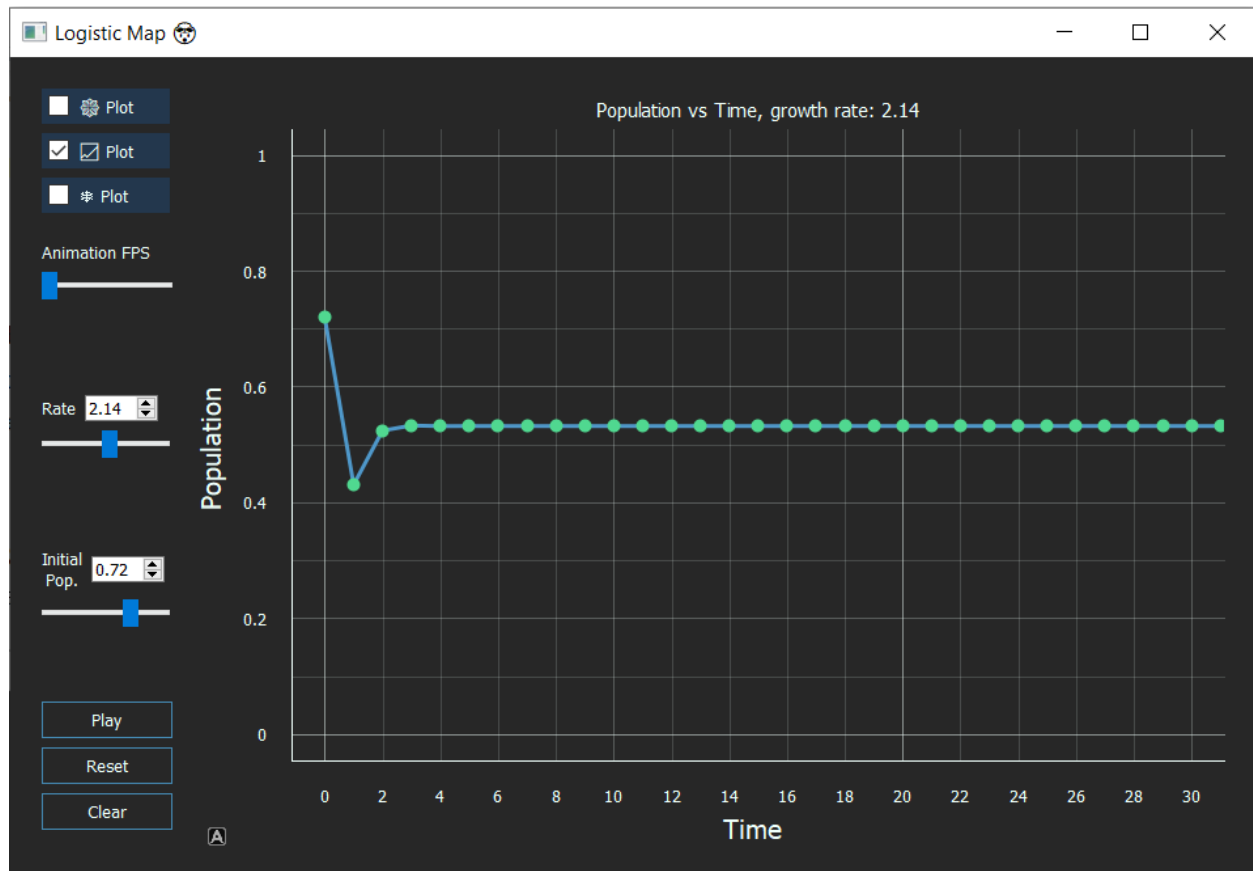
A fractal is a never-ending pattern. Fractals are infinitely complex patterns that are self-similar across different scales. They are created by repeating a simple process over and over in an ongoing feedback loop. Driven by recursion, fractals are images of dynamic systems – the pictures of Chaos. Geometrically, they exist in between our familiar dimensions. Fractal patterns are extremely familiar, since nature is full of fractals. For instance: trees, rivers, coastlines, mountains, clouds, seashells, hurricanes, etc.

Logistic Growth Equation

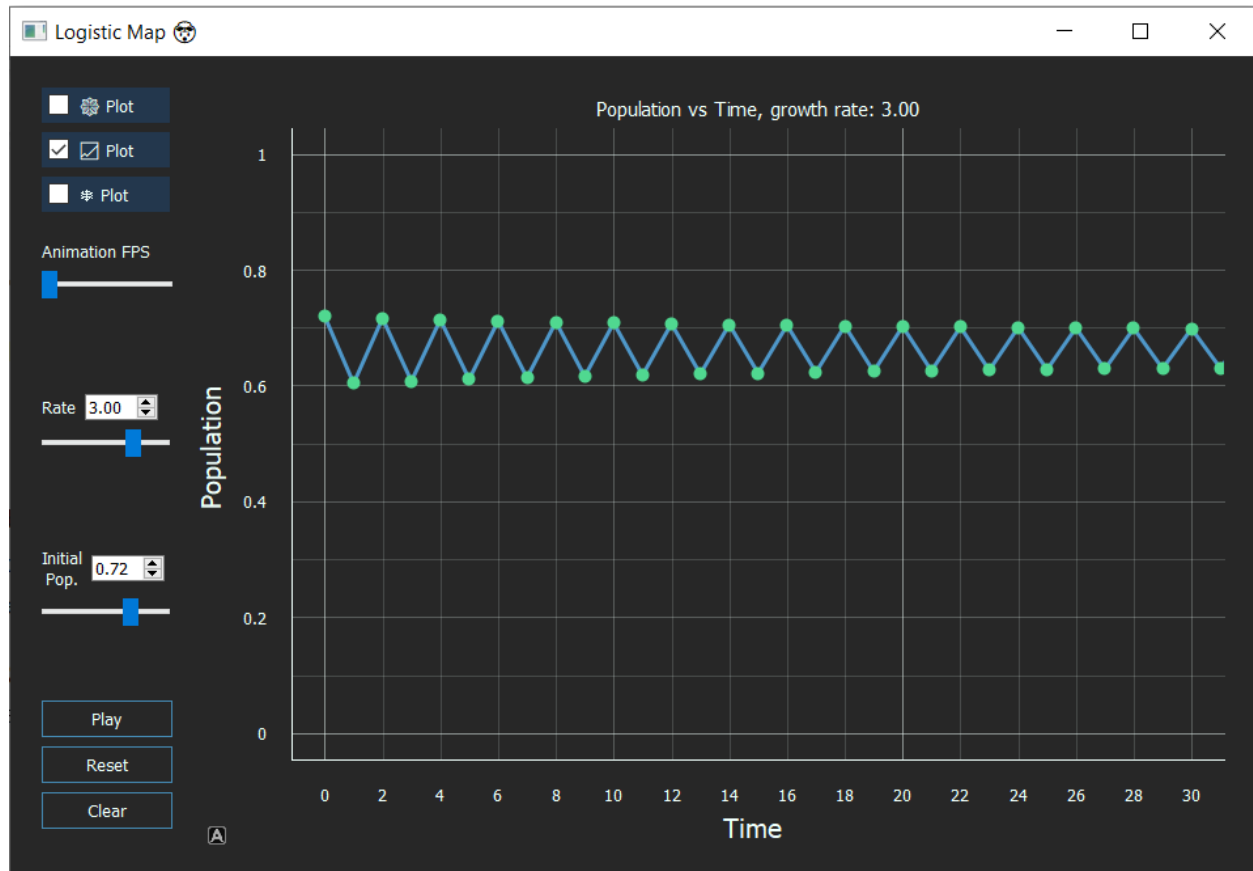
$$x_{n+1} = r \times x_n \times (1 - x_n)$$

This equation describes a logistic model for population growth in nature where r is the growth rate of the population and x is the percentage of the maximum population and hence ranges from 0 to 1.

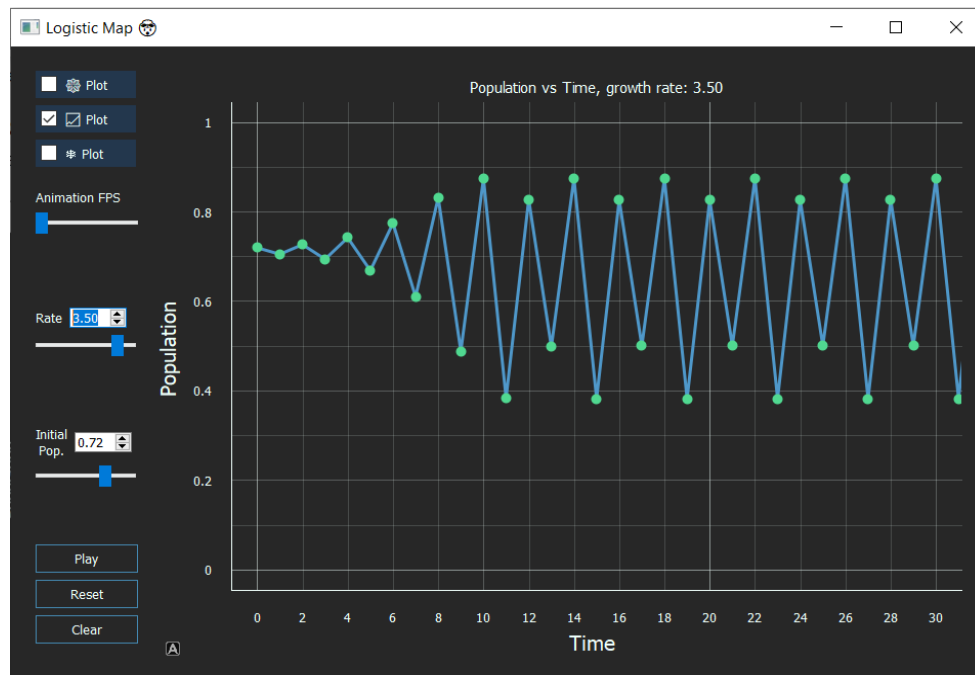
The value of population or x stabilizes to a single value for small values of the growth rate (r) and surprisingly, the stable population value does not depend on the initial population value. The stabilized population is also proportional to r .



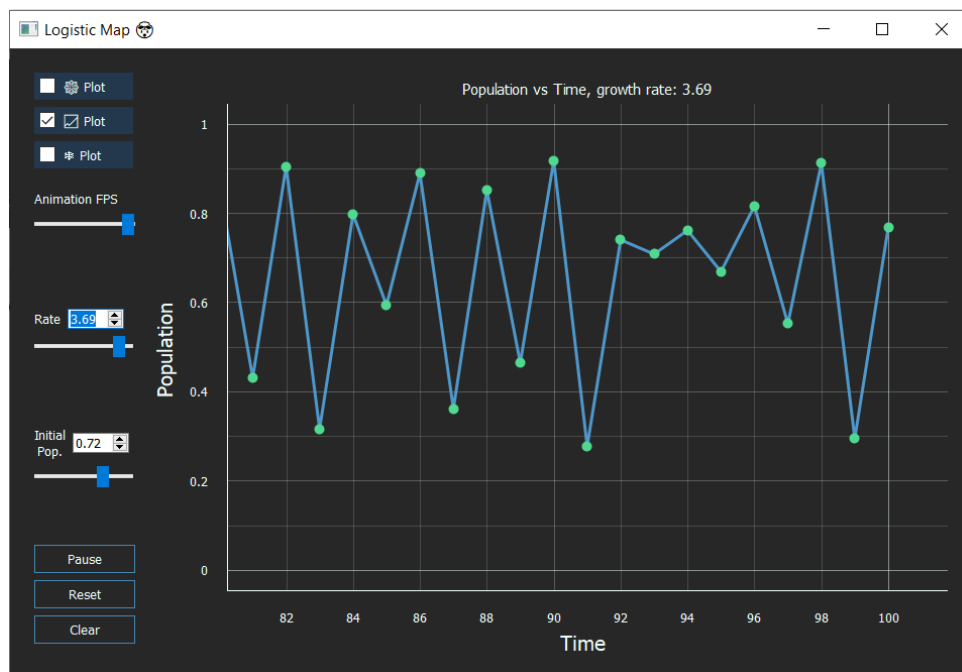
As soon as the growth rate crosses the value of 3.00, the population starts oscillating between two values and this phenomena is also termed as period doubling. This peculiar phenomena is also witnessed in nature where one year the population of a species is more and the next year the population is less. This can be seen in the Population v/s time plot below. This trend continues till a certain value of r and the difference between these 2 values increases as the value of r increases.



As the value of r increases even more, the period doubling bifurcations occur again and the population starts oscillating between 4 values.

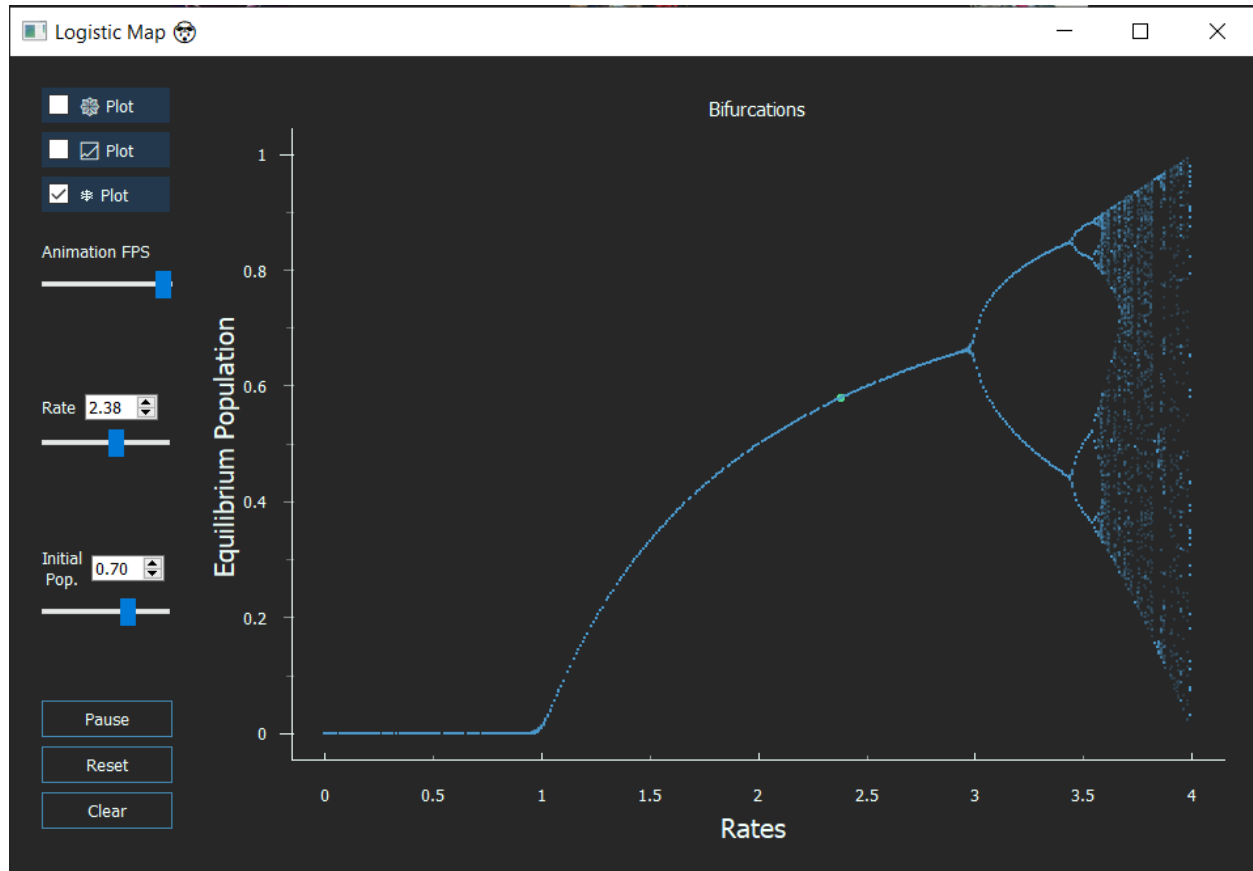


The period doubling bifurcations now starts coming faster and the population starts oscillating between 8 values, then 16 values and then slowly becomes completely random.

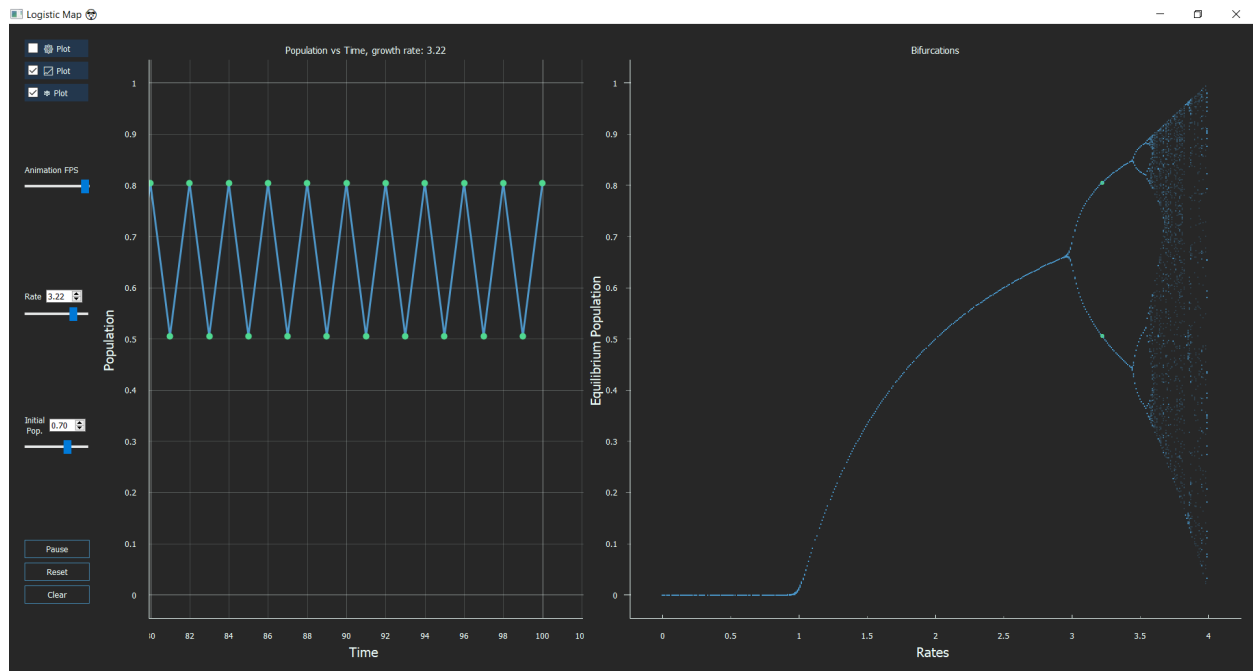


Bifurcation Diagram

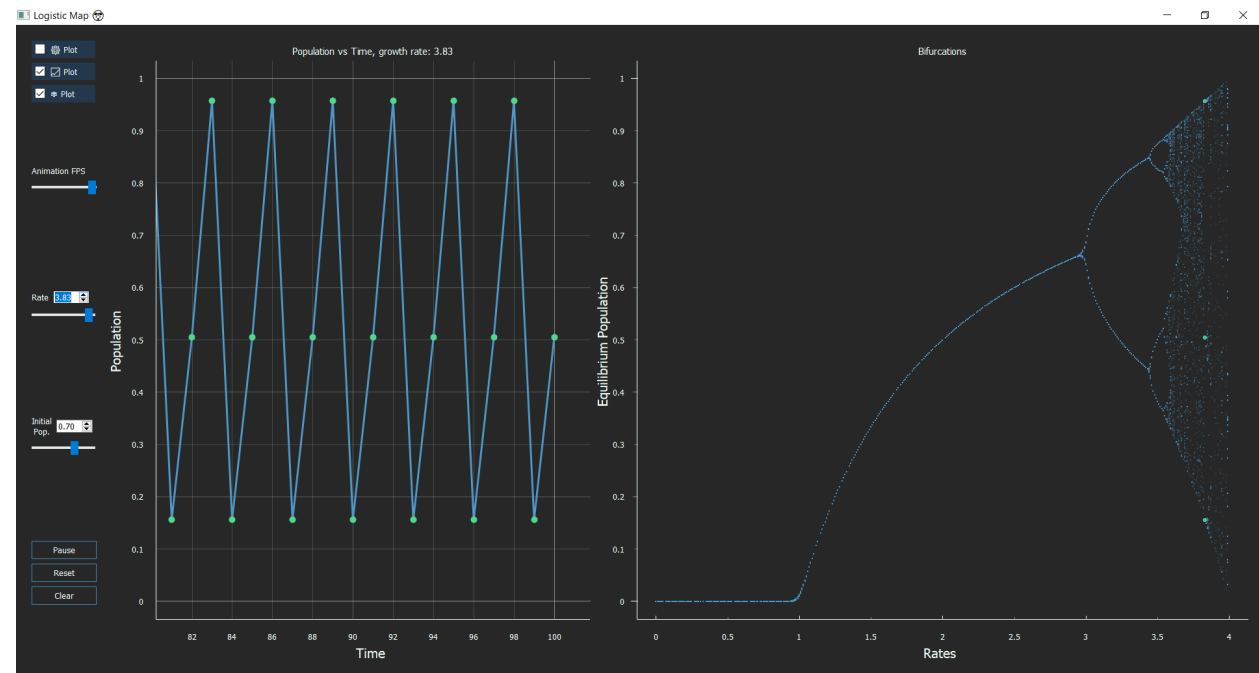
The population v/s time graph can be coupled with equilibrium population v/s rate graph to obtain the famous bifurcation diagram. The diagram shows how the equilibrium population starts oscillating between two values once the value of r crosses 3. Then it started oscillating between 4, then 8 and then at $r = 3.57$ became completely chaotic.



Comparing the two plots by taking $r = 3.22$ which is greater than 3, we can see that this particular point lies on the split branch of the bifurcation diagram, hence oscillating between 2 values. It can be seen that after splitting into 2, those 2 branches again split into 2 and this keeps on going creating a fractal like appearance.



If we closely look at a clear picture of the bifurcation diagram, we can notice some small periods of stability where the black colour goes away and white colour appears. In these small spaces, the population again starts bouncing between a fixed number of values in a certain pattern and then goes to chaos again. For example at $r = 3.83$, we get a cycle where the population oscillates between 3 different values.

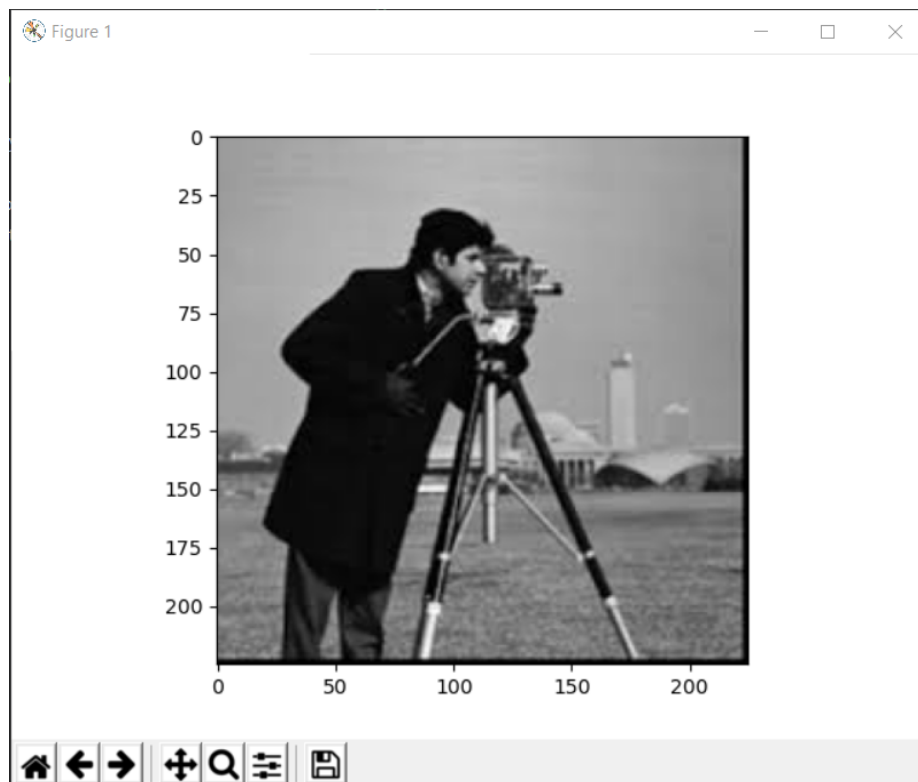


Encryption using the generated chaos

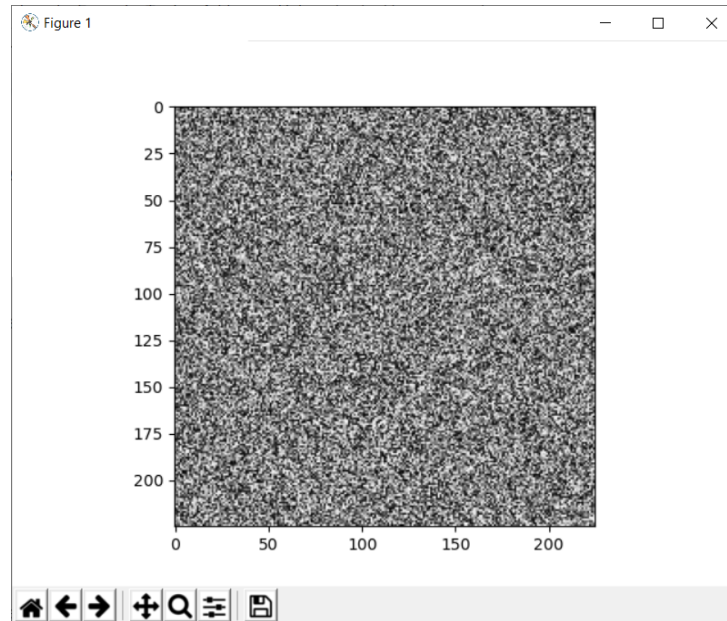
We will be using the logistic growth equation to generate pseudo-random numbers which will then be used to encrypt an image. The numbers generated will be pseudo-random in nature as they can be obtained provided we have the initial condition with us. The numbers will be used as the keys and we will be using a simple XOR operation on every pixel of the image. We will be XORing each pixel value with a pseudo-random number to obtain an encrypted pixel, which will then replace the original pixel. Following a similar fashion, we will also decrypt the image.

While decrypting the image, we will be using the same key list. If the initial conditions are altered even by a slightest amount, the key list will be changed completely and the decryption of the image will not be possible.

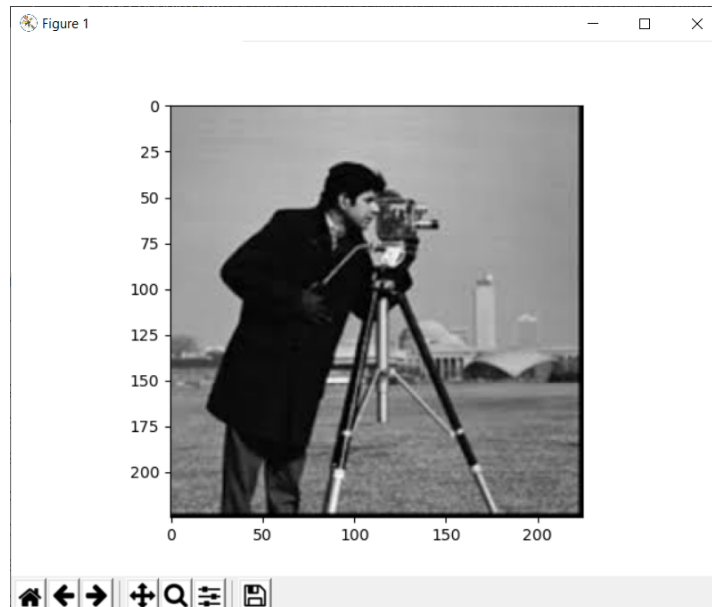
Original image



Encrypted image



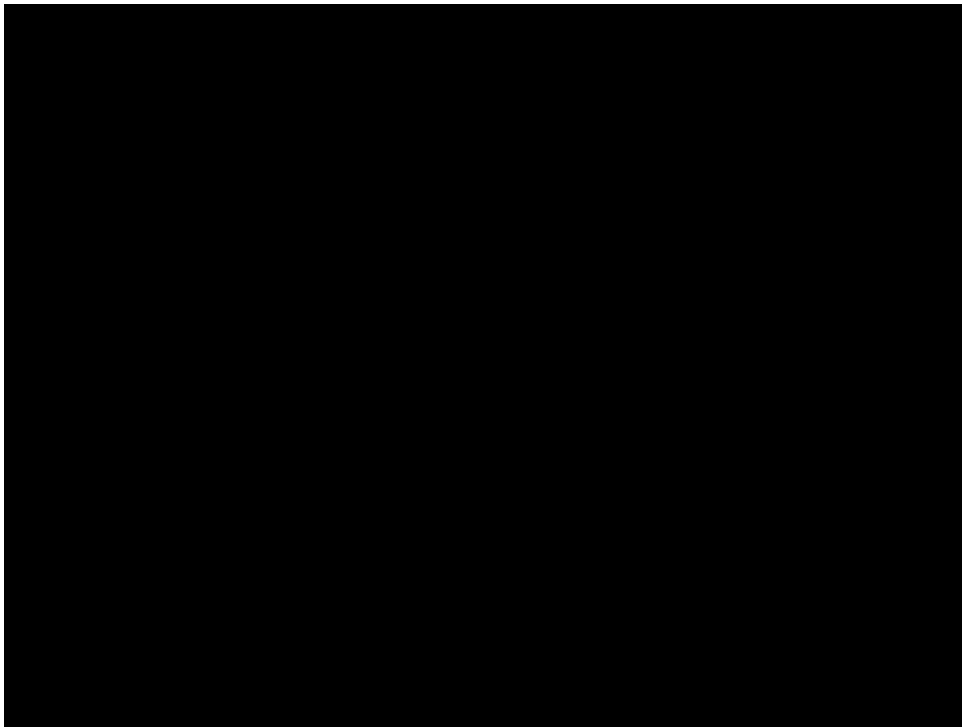
Decrypted Image



Looking for patterns in the bifurcation diagram

Zooming in

When we zoom into the bifurcation diagram, it becomes visible that there is a repeating pattern in the diagram. The diagram starts repeating and keeps on repeating till we reach the area where the chaos begins. Therefore, the bifurcation diagram is itself a **fractal!**



Connecting the bifurcation diagram to MandelBrot set

MandelBrot set

The Mandelbrot set is generated by *iteration*, which means to repeat a process over and over again. For the Mandelbrot set, the functions involved are some of the simplest imaginable: they have the form $f(x) = x^2 + c$, where c is a constant number. As we go along, we will specify exactly what value c takes.

To iterate $x^2 + c$, we begin with a *seed* for the iteration. This is a number which we write as x_0 . Applying the function $x^2 + c$ to x_0 yields the new number

$$x_1 = x_0^2 + c.$$

Now, we iterate using the result of the previous computation as the input for the next. That is

$$x_2 = x_1^2 + c$$

$$x_3 = x_2^2 + c$$

$$x_4 = x_3^2 + c$$

$$x_5 = x_4^2 + c$$

and so forth. The list of numbers x_0, x_1, x_2, \dots generated by this iteration has a name: it is called the *orbit* of x_0 under iteration of $x^2 + c$.

Let's look at some examples of the iteration of $x^2 + c$ when c is a complex number: if $c=i$, then the orbit of 0 under $x^2 + i$ is given by

$$x_0 = 0$$

$$x_1 = 0^2 + i = i$$

$$x_2 = i^2 + i = -1 + i$$

$$x_3 = (-1+i)^2 + i = -i$$

$$x_4 = (-i)^2 + i = -1 + i$$

$$x_5 = (-1+i)^2 + i = -i$$

$$x_6 = (-i)^2 + i = -1 + i$$

and we see that this orbit eventually cycles with period 2. If we change c to $2i$, then the orbit behaves very differently

$$x_0 = 0$$

$$x_1 = 0^2 + 2i = 2i$$

$$x_2 = (2i)^2 + 2i = -4 + 2i$$

$$x_3 = (-4 + 2i)^2 + 2i = 12 - 14i$$

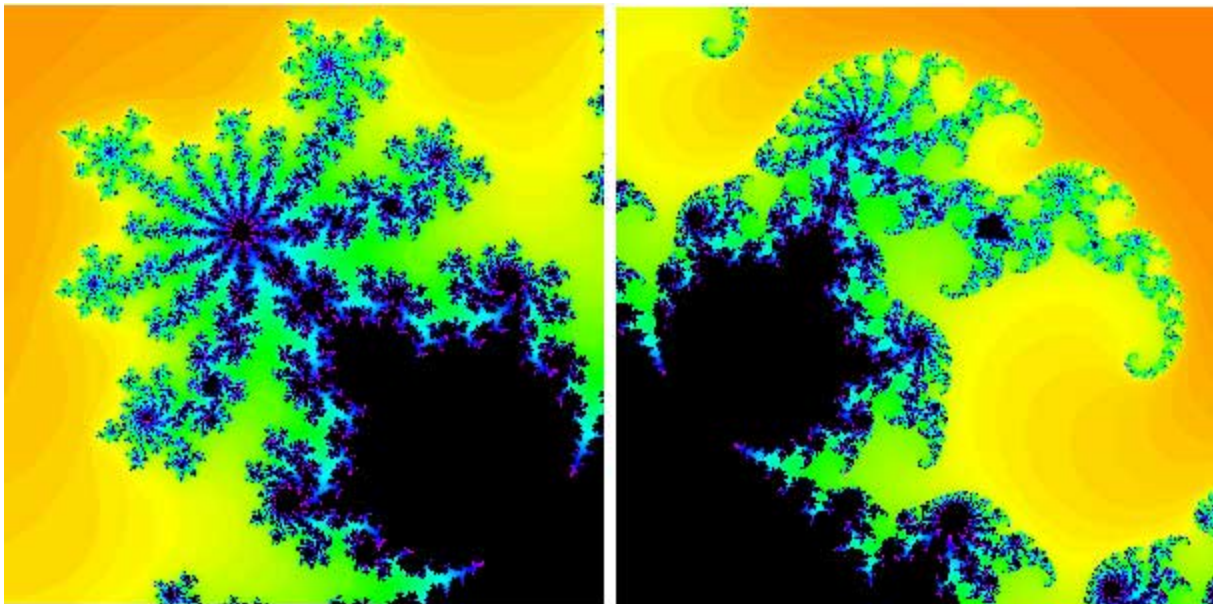
$$x_4 = (12 - 14i)^2 + 2i = -52 - 334i$$

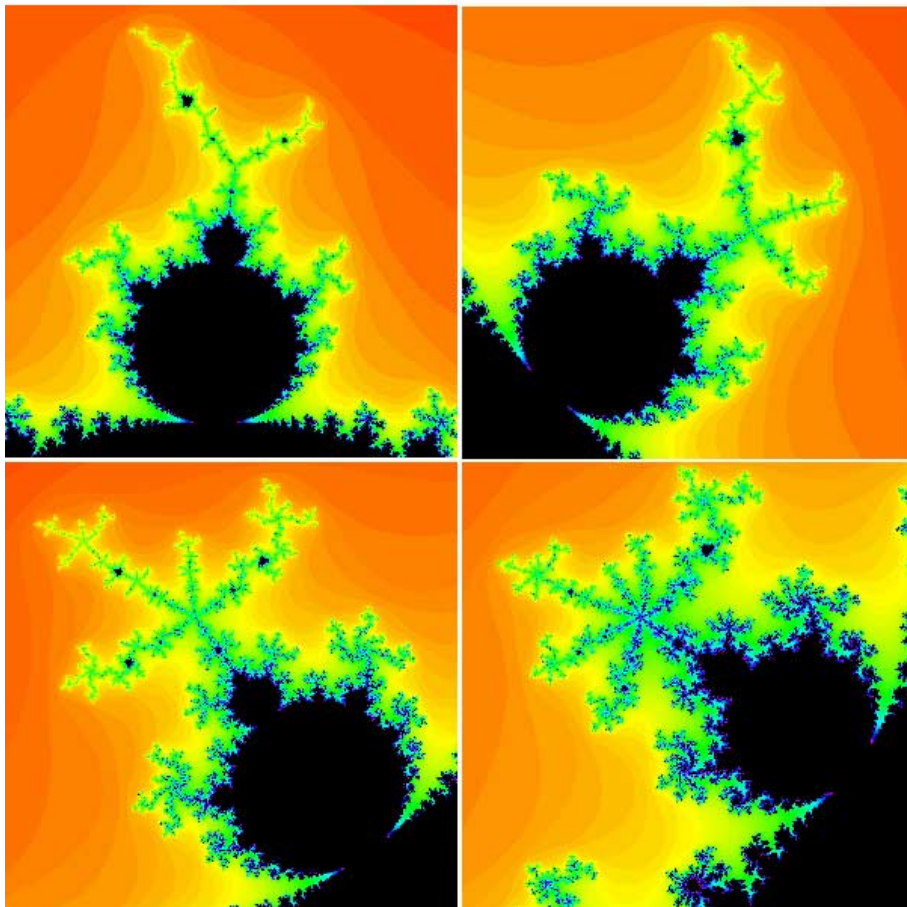
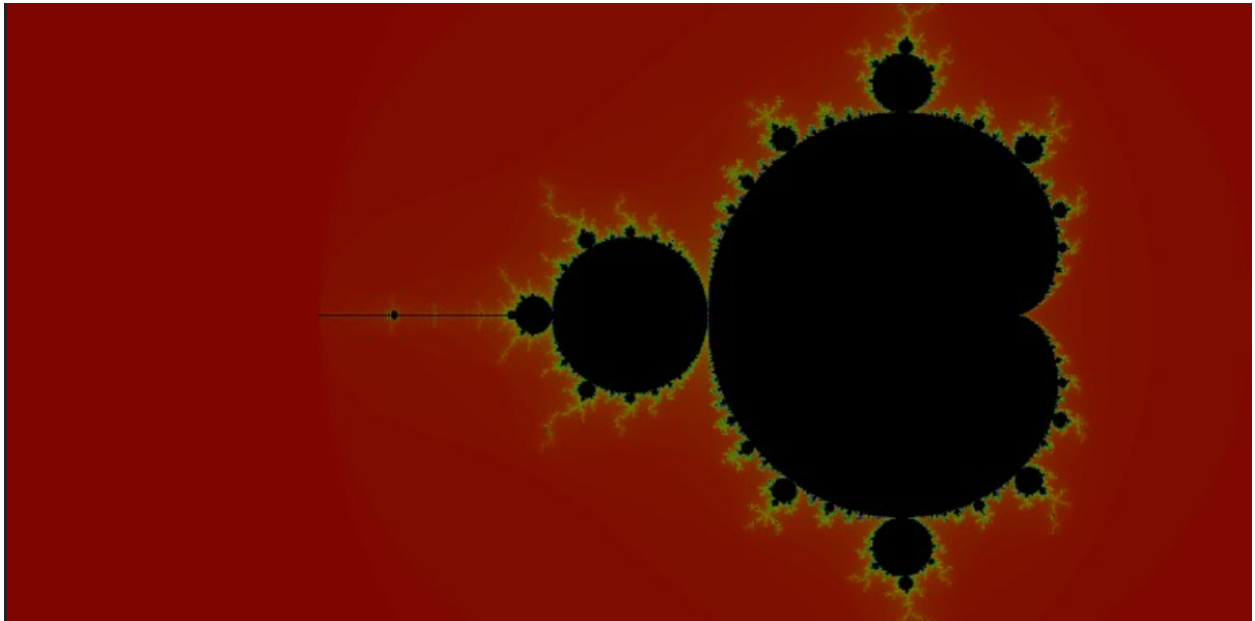
$x_5 = \text{far away from the point } 0$

$x_6 = \text{further away}$

and we see that this orbit tends to infinity in the complex plane (the numbers comprising the orbit recede further and further from the point 0, which has coordinates $(0,0)$). We make the fundamental observation that either the orbit of 0 under $x^2 + c$ tends to infinity, or it does not.

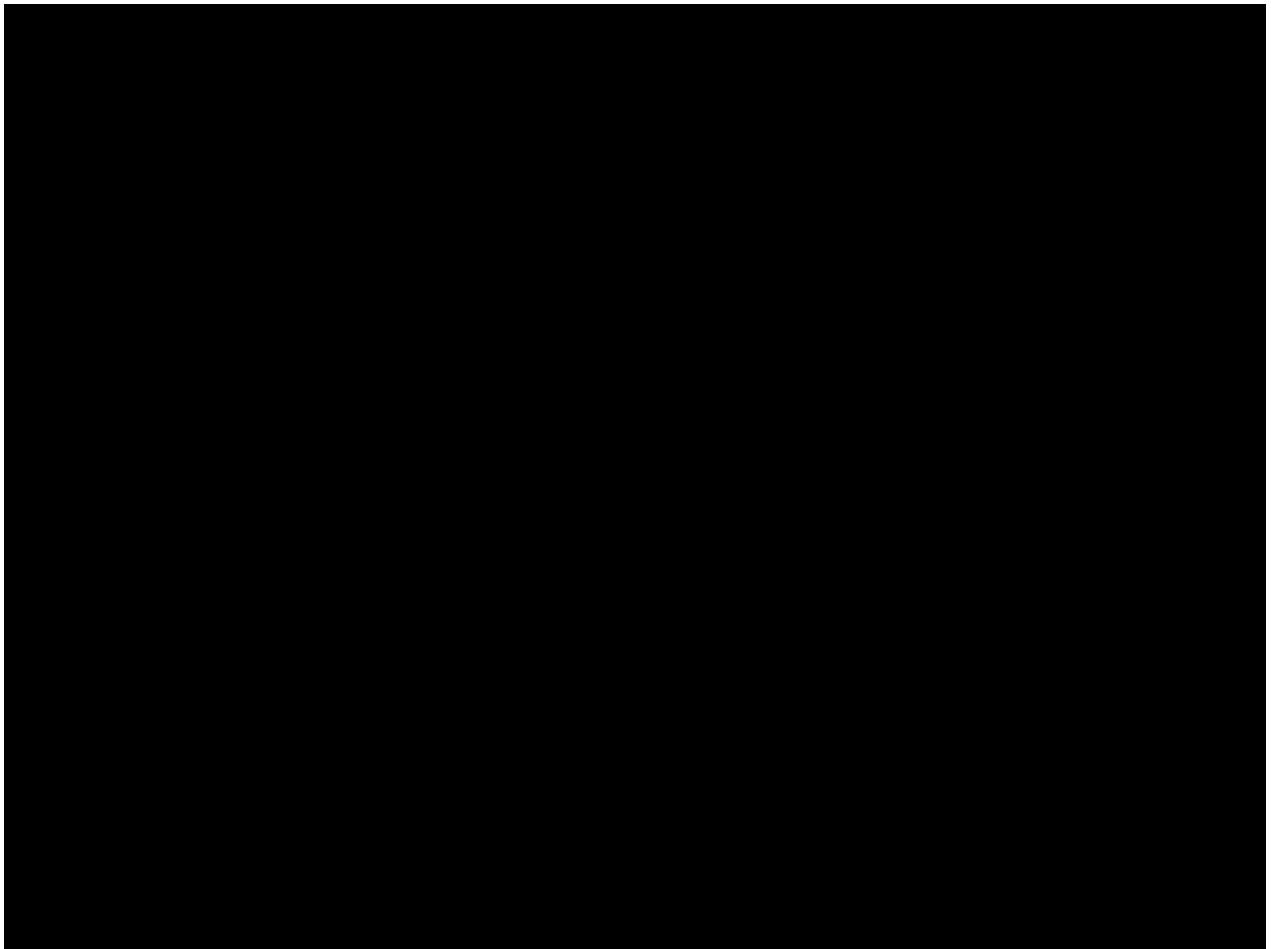
The Mandelbrot set consists of all of those (complex) c -values for which the corresponding orbit of 0 under $x^2 + c$ does not escape to infinity.





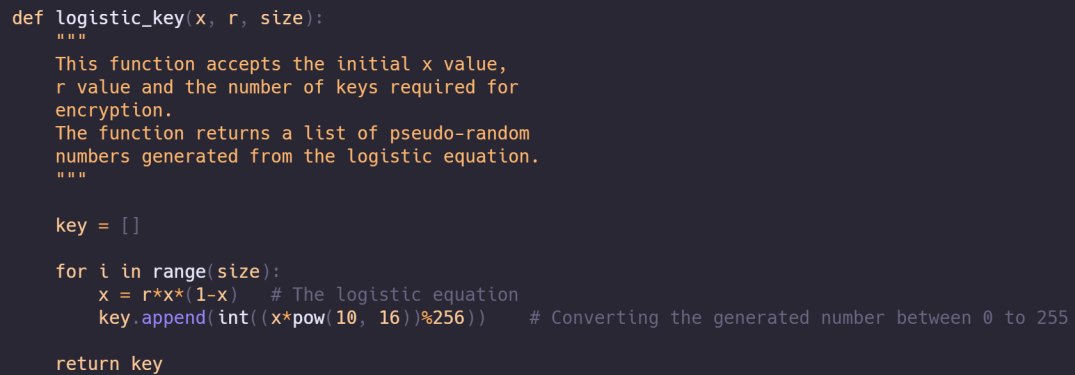
The connection

As seen in the video below, the bifurcation diagram surprisingly is a part of MandelBrot set. The main cardioid of the MandelBrot set represents the points where the population was stabilizing at a single value. The points in the main bulb oscillate between 2 values as seen in the bifurcation diagram and the points in the next bulb end up oscillating between 4 values. The needle of the MandelBrot set represents the chaotic part of the bifurcation diagram. If we zoom in on the needle, we will find another copy of the initial pattern (because it is a fractal), and this smaller version of the original pattern is the small window of stability which is amidst all the chaos.



Appendix

Code for generating keys from the logistic equation

A code block with a dark background and light-colored text, featuring a standard macOS-style window title bar with red, yellow, and green buttons at the top left. The code defines a function named 'logistic_key' that takes three arguments: 'x' (initial value), 'r' (parameter), and 'size' (number of keys). The function includes a docstring explaining its purpose and return value. It initializes an empty list 'key' and then enters a loop from 0 to 'size-1'. Inside the loop, it calculates the next value of 'x' using the logistic equation $x = r * x * (1 - x)$ and appends the integer value of $x * 10^{16} \% 256$ to the 'key' list. Finally, it returns the 'key' list.

```
def logistic_key(x, r, size):  
    """  
    This function accepts the initial x value,  
    r value and the number of keys required for  
    encryption.  
    The function returns a list of pseudo-random  
    numbers generated from the logistic equation.  
    """  
  
    key = []  
  
    for i in range(size):  
        x = r*x*(1-x) # The logistic equation  
        key.append(int((x*pow(10, 16))%256)) # Converting the generated number between 0 to 255  
  
    return key
```

Code for encrypting and decrypting an image

```
import logisticKey as key # Importing the key generating function
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as img

# Accepting an image
path = str(input('Enter the path of image\n'))
image = img.imread(path)

# Displaying the image
plt.imshow(image)
plt.show()

# Generating dimensions of the image
height = image.shape[0]
width = image.shape[1]
print(height, width)

# Generating keys
# Calling logistic_key and providing r value such that the keys are pseudo-random
# and generating a key for every pixel of the image
generatedKey = key.logistic_key(0.01, 3.95, height*width)
print(generatedKey)

# Encryption using XOR
z = 0

# Initializing the encrypted image
encryptedImage = np.zeros(shape=[height, width, 3], dtype=np.uint8)

# Substituting all the pixels in original image with nested for
for i in range(height):
    for j in range(width):
        # Using the XOR operation between image pixels and keys
        encryptedImage[i, j] = image[i, j].astype(int) ^ generatedKey[z]
        z += 1

# Displaying the encrypted image
plt.imshow(encryptedImage)
plt.show()

# Decryption using XOR
z = 0

# Initializing the decrypted image
decryptedImage = np.zeros(shape=[height, width, 3], dtype=np.uint8)

# Substituting all the pixels in encrypted image with nested for
for i in range(height):
    for j in range(width):
        # Using the XOR operation between encrypted image pixels and keys
        decryptedImage[i, j] = encryptedImage[i, j].astype(int) ^ generatedKey[z]
        z += 1

# Displaying the decrypted image
plt.imshow(decryptedImage)
plt.show()
```

References

1. <https://github.com/jonnyhyman/Chaos>
2. <https://www.youtube.com/watch?v=ovJcsL7vyrk>