Universidad de Granada

Facultad de Ciencias

trabajo de fin de grado

La ecuación logística discreta y sus aplicaciones a criptografía

grado en matemáticas

Presentado por:
Nadia Moya Díaz Santos

Tutor:
Maria José Cáceres Granados
Departamento de Matemática Aplicada.

Curso académico 2022-2023

La ecuación logística discreta y sus aplicaciones a criptografía.

Nadia Moya Díaz Santos

Nadia Moya Díaz Santos. *La ecuación logística discreta y sus aplicaciones a criptografía.*

Trabajo de fin de Grado. Curso académico 2022-2023.

Responsable de tutorización: Maria José Cáceres Granados
Departamento de Matemática Aplicada.

Grado en Matemáticas
Facultad de Ciencias
Universidad de Granada

Declaración de originalidad

D./Dña. Nadia Moya Díaz Santos

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2022-2023, es original, entendida esta, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 20 de junio de 2023

Fdo: Nadia Moya Díaz Santos

Índice

Summary

The aim of this project is to analyse the relationship between two disciplines that at first sight appear to be different: chaotic equations and cryptography. To do so, we are going to study how chaotic equations can influence the design and security of cryptographic algorithms. Generally, these algorithms are based on the generation of pseudo-random sequences that are difficult to decrypt without the right key, but can be predictable. Thanks to the high randomness offered by chaotic systems, we will see that they can provide a more solid basis in the cryptographic area.

This work consists of three chapters which are developed as follows: In the first chapter, we have dealt with the study of difference equations. First, we have introduced the difference equations by establishing their general form and providing some specific examples such as the first order difference equations and the Fibonacci sequence. The well-known direct solution method, known as the method of indeterminate coefficients, has also been explained. In the following, we will present different graphical methods to describe the solutions of the difference equations when it is practically impossible to obtain them analytically. We will present graphs that illustrate the behaviour of the solutions and explain the so-called Cobweb method. The chapter also deals with the stability analysis of the solutions of difference equations. We study the concept of stability, and we present criteria and theorems that allow us to determine the stability of the solutions. Finally, we conclude the chapter by discussing bifurcations in difference equations that depend on a parameter.

The next chapter focuses on the basis of our work, the discrete logistic equation. The main goal of this chapter is to understand its properties and behaviour. First, we will introduce the discrete logistic equation and see what conditions it must fulfil in order to make biological sense, since this equation is used to model the evolution of the population of a given species. We will then study the characteristics and properties of the discrete logistic equation. The chapter also deals with the analysis of the stability of the solutions of this equation, where we will use the methods explained in chapter one to determine the stability as a function of the parameters of the model. In addition, we will present graphs that illustrate the behaviour of the discrete logistic equation in different cases. These practical examples will help to understand how the population evolves over time and how it is affected by changes in the parameters. We will conclude that it is a chaotic equation since for certain values of the parameters chaotic solutions appear.

In the third chapter, we relate the discrete logistic equation to cryptography by implementing Python program for the encryption and decryption of an image. This program uses the discrete logistic equation to generate 3 chaotic orbits that will be used to reorganise the data in the following way: the first orbit will be in charge of mixing the original information, the second one will be combined with the mixed information from the previous step thus increasing the size of the vector that will contain the encrypted information and the last one will be used for further confusion of the data. In the chapter we will explain the steps in detail and show the program that will encrypt and decrypt the image. Finally, the chapter will conclude with the contribution of a new code to considerably reduce the compilation time of the first one.

Finally, in the last chapter we have provided some activities to be carried out in secondary education in relation to the chaos, provided by the discrete logistic equation, and cryptography.

In summary, this project focuses on the study of the chaotic behaviour of the discrete logistic equation and its application in the field of cryptography. Chaotic equations offer an interesting theoretical and practical basis for cryptography, providing pseudo-random sequences and complexity necessary to guarantee the security of the information being transmitted. The relationship between the two disciplines is an active field of research and development in the field of information security. By exploring the characteristics of the discrete logistic equation and its relationship with chaos, we hope to contribute to the generation of pseudo-random sequences that are more secure and more difficult for attackers to crack.

Introducción

Desde hace mucho tiempo, la criptografía ha jugado un papel fundamental en la seguridad de la información y en la confidencialidad de los datos. Hoy en día vivimos en un mundo interconectado, por lo que esta rama ha ido ganando aún más relevancia. Por su parte, las ecuaciones caóticas muestran comportamientos impredecibles y altamente sensibles a las condiciones iniciales.

En los últimos años, la criptografía y el comportamiento caótico de las ecuaciones han captado la atención de quienes trabajan en la seguridad de la información. Es por ello que se ha comenzado a investigar la opción de aprovechar las propiedades de las ecuaciones caóticas para reforzar los algoritmos criptográficos.

Este trabajo tiene como objetivo mostrar el uso del comportamiento caótico de la ecuación logística discreta en la criptografía, estudiando de qué manera las soluciones caóticas pueden influir en el diseño y la seguridad de los algoritmos criptográficos. Normalmente, estos algoritmos se basan en la generación de secuencias pseudoaleatorias que son difíciles de descifrar sin la clave adecuada, pero que a su vez pueden llegar a ser predecibles. Gracias a la gran aleatoriedad que ofrecen los sistemas caóticos, éstos pueden proporcionar una base más sólida en la criptografía.

Concretamente a lo largo de este trabajo estudiaremos, analizando la referencia [13], un algoritmo criptográfico basado en la ecuación logística discreta. Para ello, comenzaremos el primer capítulo introduciendo, con el apoyo de [10], las ecuaciones en diferencias, dando ejemplos de éstas y respondiendo, mediante teoremas y proposiciones, a preguntas acerca del comportamiento de sus soluciones. Tras esto, nos centraremos en la base de nuestro trabajo, la ecuación logística discreta. Veremos que esta ecuación se trata de una ecuación en diferencias no lineal de primer orden y que surge de la necesidad de mejorar una ecuación que modeliza la evolución de la población de una determinada especie, la ecuación de Malthus.

A lo largo del capítulo dos nuestro objetivo será desarrollar y estudiar en profundidad la ecuación logística discreta, analizando sus propiedades y características y explorando como pequeños cambios en sus condiciones iniciales pueden llegar a tener un impacto significativo en su comportamiento. Cabe destacar que el estudio de esta ecuación formaba parte de la asignatura Modelos Matemáticos I que se imparte en el grado, pero durante el desarrollo de ésta, esa parte no pudo ser vista, por lo tanto hemos tenido que indagar aún más en este tema.

El tercer capítulo, tratará la implementación en Python de un algoritmo descrito en el artículo [13]. Éste se encargará de la encriptación y desencriptación de una imagen. Para su implementación haremos uso de la ecuación logística discreta dando lugar a tres órbitas caóticas que nos servirán para reorganizar los datos de la siguiente manera: la primera órbita será la encargada de mezclar la información original, la segunda se combinará con la información mezclada del paso anterior aumentando así el tamaño del vector que contendrá la información cifrada y, por último, la tercera se utilizará para una mayor confusión de los datos. Tras la compilación de este algoritmo, concluimos que se ha desarrollado con éxito, pero muy lentamente. Es por ello por lo que al final del trabajo hemos propuesto un nuevo método para mejorar el algoritmo propuesto en [13], concluyendo que este nuevo algoritmo es mucho más rápido y además la imagen encriptada queda mucho más homogénea, de modo que debería ser más difícil de desencriptar en caso de atacante.

Finalmente el trabajo concluye, cumpliendo así todos los objetivos matemáticos planteados en este Trabajo de Fin de Grado, con la aportación de algunas propuestas o ideas de actividades para ESO y/o Bachillerato, con el fin de captar la atención del alumnado en el ámbito de las matemáticas y facilitar la comprensión del comportamiento de la ecuación logística discreta.

En resumen, este Trabajo de Fin de Grado se centra en el estudio del comportamiento caótico de la ecuación logística discreta y su aplicación en el ámbito de la criptografía. Al explorar las características de esta ecuación y su relación con el caos, se espera contribuir en la generación de secuencias pseudoaleatorias más seguras y difíciles de descifrar para quien quiera atacar.

1. Capítulo 1: Ecuaciones en diferencias.

En este primer capítulo introduciremos algunos conceptos básicos acerca de ecuaciones en diferencias. Comenzaremos con una breve introducción, ayudándonos de [1], para diferenciar los sistemas dinámicos discretos de los continuos. Tras esto, discutiremos qué es una ecuación en diferencias y cómo son sus soluciones. También, comentaremos algunos conceptos básicos en el estudio de este tipo de ecuaciones como son las órbitas, los ciclos y las soluciones constantes, estudiando además la estabilidad de estos dos últimos. Finalmente, concluiremos el capítulo hablando acerca de bifurcaciones en ecuaciones en diferencias que dependen de un parámetro. Para el desarrollo de esta sección, nos hemos basado fundamentalmente en el libro [10]. Todos estos conceptos son elementales para estudiar, como veremos en el siguiente capítulo, la ecuación logística discreta.

Los sistemas dinámicos son los encargados de estudiar la evolución en el tiempo de determinados fenómenos. Dependiendo de si el tiempo es considerado una variable discreta o continua, éstos se clasifican en sistemas dinámicos discretos o sistemas dinámicos continuos. Los primeros son los que nos encargaremos de estudiar en este capítulo, mediante las ecuaciones en diferencias. Mientras que los segundos son modelados por ecuaciones diferenciales.

Definición 1.1. Una *ecuación en diferencias* es una expresión de la forma:

$$x_{k+1} = f(k, x_k), \quad k = 0, 1, 2, ... \tag{1.1}$$

siendo $f : \mathbb{N} \cup \{0\}$ x $\mathbb{R} \longrightarrow \mathbb{R}$ una aplicación continua que depende del entero no negativo k, el cual indica los sucesivos periodos.

En este trabajo nos centraremos, como veremos más adelante, en los casos particulares donde $x_{k+1} = f(x_k)$, ya que la ecuación logística discreta es una ecuación en diferencias de este tipo.

Para entender mejor las ecuaciones en diferencias, veamos a continuación algunos ejemplos específicos de éstas:

Ejemplo 1.1. Ecuaciones lineales en diferencias de primer orden.

Son ecuaciones en diferencias de la forma :

$$x_{k+1} = a(k)x_k + b(k), \tag{1.2}$$

donde $a(k)$ y $b(k)$ representan dos funciones reales que están definidas para los valores enteros $k = 0, 1, 2, 3, ...$

La solución general de la ecuación (1.2) viene dada por la expresión $x_k = x_h + x_p$, donde x_h representa la solución general de la ecuación lineal homogénea $x_{k+1} = a(k)x_k$ y, x_p se trata de una solución particular cualquiera de la ecuación completa (1.2).

En primer lugar, consideramos la ecuación homogénea:

$$x_{k+1} = a(k)x_k. \tag{1.3}$$

La solución de ésta, que comienza en el valor inicial x_0, la obtenemos aplicando el proceso iterativo dado por (1.3), obteniendo así:

$$x_1 = a(0)x_0,$$

$$x_2 = a(1)x_1 = a(0)a(1)x_0,$$

$$x_3 = a(2)x_2 = a(0)a(1)a(2)x_0,$$

Aplicando ahora inducción, llegamos a que la solución de la ecuación homogénea, con dato inicial x_0, es de la forma:

$$x_h = x_0 \prod_{i=0}^{k-1} a(i). \tag{1.4}$$

En el caso en el que x_0 variara en $\mathbb{R}$ en la fórmula (1.4), obtendríamos la expresión de la solución general de (1.3). Y vendría dada de la siguiente forma:

$$x_h = C \prod_{i=0}^{k-1} a(i), \quad C \in \mathbb{R}. \tag{1.5}$$

Una vez visto esto, solo faltaría encontrar una solución particular x_p. Generalmente, esta solución se obtiene aplicando el *método de variación de constantes*, que consiste en determinar $C(k)$ para que

$$x_p(k) = C(k) \prod_{i=0}^{k-1} a(i) \tag{1.6}$$

sea solución de mi ecuación completa (1.2).

Efectivamente, si (1.6) es solución de mi ecuación completa (1.2) tenemos que :

$$C(k+1) \prod_{i=0}^{k} a(i) = a(k)C(k) \prod_{i=0}^{k-1} a(i) + b(k) = C(k) \prod_{i=0}^{k} a(i) + b(k).$$

Suponiendo ahora que $a(k) \neq 0$, $\forall k = 0, 1, 2, ...$ podemos dividir por $\prod_{i=0}^{k} a(i)$:

$$C(k+1) = C(k) + (\prod_{i=0}^{k} a(i))^{-1} b(k), \tag{1.7}$$

Aplicando ahora sucesivas veces el proceso iterativo dado por (1.7), se obtiene:

$$C(1) = C(0) + \frac{b(0)}{a(0)},$$

$$C(2) = C(1) + \frac{b(1)}{a(0)a(1)} = C(0) + \frac{b(0)}{a(0)} + \frac{b(1)}{a(0)a(1)},$$

y, en general, por inducción:

$$C(k) = C(0) + \sum_{j=0}^{k-1} b(j) \prod_{i=0}^{j} a(i)^{-1}. \tag{1.8}$$

Como solo necesitamos una solución particular, podemos imponer $C(0) = 0$ y, por tanto, sustituyendo el valor de $C(k)$, que nos da la ecuación (1.8), en la ecuación (1.6) tenemos que nuestra solución particular vendrá dada por la siguiente expresión:

$$x_p(k) = \sum_{j=0}^{k-1} b(j) \prod_{i=j+1}^{k-1} a(i). \tag{1.9}$$

Podemos concluir entonces que la solución general de la ecuación completa (1.2) es de la forma:

$$x_k = x_h + x_p = C \prod_{i=0}^{k-1} a(i) + \sum_{j=0}^{k-1} b(j) \prod_{i=j+1}^{k-1} a(i), \quad C \in \mathbb{R}. \tag{1.10}$$

Ejemplo 1.2. Vamos a ver un caso concreto del Ejemplo 1.1.

Determinemos, cuando $x_0 = 0$, la solución de la ecuación:

$$x_{k+1} = (k+3)x_k + (k+4)!$$

Aplicando el procedimiento anterior sabemos que tenemos que encontrar la solución general de la ecuación homogénea y una solución particular.
La solución general de la ecuación homogénea viene dada, como ya hemos visto, por la siguiente expresión:

$$x_h(k) = C\prod_{i=0}^{k-1}(i+3) = \frac{C}{2}(k+2)!$$

A continuación debemos buscar, aplicando el método de variación de constantes, una solución particular de la forma :

$$x_p(k) = \frac{C(k)}{2}(k+2)!$$

Para ello, sustituimos la expresión anterior en la ecuación de partida:

$$\frac{C(k+1)}{2}(k+3)! = (k+3)\frac{C(k)}{2}(k+2)! + (k+4)! = \frac{C(k)}{2}(k+3)! + (k+4)!$$

y, posteriormente, dividimos ambos miembros por $\frac{(k+3)!}{2}$:

$$C(k+1) = C(k) + 2(k+4).$$

Imponiendo $C(0) = 0$, tenemos que el valor de $C(k)$ es:

$$C(k) = \sum_{j=0}^{k-1} 2(j+4) = 2\sum_{j=0}^{k-1}(j+4).$$

Observamos que se trata de la suma de los k primeros términos de la progresión aritmética $j+4$ y, por tanto, aplicando la siguiente fórmula:

$$S_k = \tfrac{k}{2}(\text{primer término} + \text{último término}),$$

que nos da el valor de los k términos consecutivos de una progresión aritmética, llegamos a que:

$$C(k) = (4 + (k+3))k = (k+7)k.$$

Por tanto, la solución general de mi ecuación completa es:

$$x_k = x_h + x_p = \tfrac{C}{2}(k+2)! + \tfrac{(k+7)k}{2}(k+2)! = \tfrac{(k+2)!}{2}(k^2+7k+C).$$

Aplicando por último la condición de que $x_0 = 0$ obtenemos que $C = 0$.

En consecuencia, concluimos finalmente que la solución será $x_k = \frac{(k+2)!}{2}(k^2+7k)$.

Ejemplo 1.3. Ecuación de *Malthus*.

Para el desarrollo de este ejemplo hemos tenido en cuenta los apuntes impartidos en la asignatura de Modelos Matemáticos I, [15].

La ecuación de Malthus se trata de una ecuación lineal homogénea que modeliza la evolución de la población de una determinada especie aislada de la siguiente manera:

En primer lugar, llamamos p_k al número de individuos en el instante temporal k, α_n a la tasa de natalidad por individuo y α_m a la tasa de mortalidad por individuo. Entonces, $\alpha_n.x_k$ es el número de nacimientos en el periodo k y $\alpha_m.x_k$ el número de muertes en dicho periodo.

En consecuencia, la ecuación que nos indica el número de individuos en el periodo de tiempo *k+1*, viene dada por la siguiente expresión:

$$p_{k+1} = (1 + \alpha_n - \alpha_m)p_k, \qquad k = 0, 1, 2, 3, ... \tag{1.11}$$

Esta ecuación en diferencias de primer orden es la llamada ecuación de *Malthus*.

En esta ecuación, se denomina *razón de crecimiento* R al valor $R = 1 + \alpha_n - \alpha_m$, cumpliendo así que $R = \frac{p_{k+1}}{p_k}$.

Por otro lado, la *tasa de crecimiento* viene dada por $\alpha = \alpha_n - \alpha_m$. Es fácil ver que $\alpha \geq -1$,ya que la tasa de natalidad por individuo $\alpha_n \geq 0$ y la tasa de mortalidad por individuo $1 \geq \alpha_m \geq 0$. Además, podemos relacionar la *tasa de crecimiento* con la *razón de crecimiento* de la siguiente manera: $\alpha = R - 1 = \frac{p_{k+1} - p_k}{p_k}$

Tras estos dos nuevos conceptos, podemos reescribir la *ecuación de Malthus* en función de ellos de la siguiente manera:

$$p_{k+1} = (1 + \alpha)p_k = Rp_k, \qquad k = 0, 1, 2, 3, ... \tag{1.12}$$

Además, como hemos dicho anteriormente, nos interesa conocer la evolución en el tiempo de la población. Para ello, tendremos en cuenta la expresión general de las soluciones de una ecuación lineal homogénea. Sabemos, por lo explicado en el Ejemplo 1.1, que estas soluciones tienen la forma $p_h = C \prod_{i=0}^{k-1} a(i), \quad C \in \mathbb{R}$. En el caso de la ecuación de Malthus el término $a(i) = 1 + \alpha$, luego observamos que dicha evolución dependerá del valor de α, tal que:

- Si $\alpha > 0$, la población crecerá sin límite.
- Si $-1 \leq \alpha < 0$, la población irá decreciendo hasta llegar a la extinción.
- Si $\alpha = 0$, la población se mantendrá constante.

Cabe destacar que este modelo presenta algunas complicaciones ya que supone que las poblaciones se reproducen de forma ideal, es decir, sin que exista ningún factor que limite su crecimiento. Un modelo un poco más realista es el de la ecuación logística discreta, en el que nos centraremos más adelante.

Ejemplo 1.4. Sucesión de *Fibonacci*.

Aunque estemos estudiando las ecuaciones en diferencias de primero orden, también existen ecuaciones en diferencias de orden superior. Un ejemplo muy conocido y destacado es la sucesión de Fibonacci. Esta sucesión se trata de una ecuación en diferencias de segundo orden que viene dada por la siguiente expresión:

$$f_{k+2} = f_{k+1} + f_k, \quad k = 0, 1, 2, ..., \quad f_0 = f_1 = 1.$$

Obteniendo así la sucesión $\{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...\}$. Además, cabe observar que para arrancar la sucesión hemos necesitado dos valores iniciales. Esto se debe a que, como ya hemos mencionado, se trata de una ecuación en diferencias de segundo orden. En general, para una ecuación en diferencias de orden k se necesitan k condiciones iniciales.

Una vez vistos y analizados estos ejemplos vamos a centrarnos en las soluciones de las ecuaciones en diferencias. Generalmente, la obtención de soluciones para ecuaciones en diferencias no lineales suele ser un trabajo prácticamente imposible, salvo en casos muy particulares. Por ello, en lugar de obtener una expresión analítica de las soluciones, normalmente se estudian ciertas propiedades de éstas, así como su comportamiento asintótico sin necesidad de llegar a calcularlas explícitamente.

1.1. Órbitas.

Para elaborar esta sección hemos tenido de nuevo en cuenta los apuntes impartidos en la asignatura de Modelos Matemáticos I [15], además del libro [10].

Como ya hemos mencionado anteriormente, nos vamos a centrar en las ecuaciones en diferencias de primer orden del tipo $x_{k+1} = f(x_k)$, donde $f : \mathbb{R} \longrightarrow \mathbb{R}$ es una función continua.

Definición 1.2. Una sucesión de números reales $\{x_k\}$ diremos que es una *solución* de la ecuación (1.1) si verifica que entre cualquier término de la sucesión y el siguiente, se cumple dicha igualdad.

En particular, diremos que una solución es *constante* si se trata de una sucesión de la forma $x_k = \{c, c, c, ..., c, ...\}$, donde c es un punto fijo de f, es decir, $f(c) = c$.

Por otro lado, diremos que una solución es un *ciclo de orden p* o un *p-ciclo* si es una sucesión de la forma $x_k = \{c_0, c_1, ..., c_{p-1}, c_0, c_1, , ..., c_{p-1}, ...\}$, siendo $c_0, c_1, ..., c_{p-1}$ p valores distintos entre sí, que verifican:

$$c_1 = f(c_0),\ c_2 = f(c_1),...,\ c_{p-1} = f(c_{p-2}),\ c_0 = f(c_{p-1}).$$

Por tanto, un ciclo de orden p verfica que $f^p(c_0) = c_0$, es decir, c_0 es un punto fijo de f^p.

Ejemplo 1.5. Si partimos de la ecuación en diferencias $x_{k+1} = -x_k$, $k \geq 0$, sus soluciones son de la forma $x_k = (-1)^k x_0$. Por tanto, se trata de ciclos de orden 2: $\{x_0, -x_0, x_0, -x_0, ...\}$

Tras estas definiciones, observamos que cuando nuestro sistema parte de un estado inicial x_0, su evolución en el tiempo corresponde a la sucesión $x_0, x_1, x_2, ...$ Obteniendo de forma recursiva $x_1 = f(x_0), x_2 = f(x_1) = f^2(x_0)$, hasta llegar a la expresión general: $x_k = f^k(x_0)$.

Definición 1.1.1. Llamamos *órbita* o *trayectoria* de la ecuación en diferencias (1.1) al conjunto de valores $\{x_0, x_1, x_2, ...x_k, ...\}$=$\{x_0, f(x_0), f^2(x_0), ..., f^k(x_0), ...\}$ que toma la sucesión solución de la ecuación (1.1), con dato inicial x_0.

Definición 1.1.2. El conjunto formado por todas las órbitas asociadas a la ecuación en diferencias es el llamado *retrato de fases*.

Particularmente, cabe destacar dos tipos de órbitas, las órbitas estacionarias y las periódicas. Las órbitas estacionarias son las órbitas de las soluciones constantes y se caracterizan por tener un único punto. En efecto, si $x_k = c$ es una solución constante, entonces:

$$x_0 = c, x_1 = f(x_0) = f(c) = c, x_2 = f(x_1) = f(c) = c, ..., x_k = f(x_{k-1}) = f(c) = c$$

Por otro lado, una órbita diremos que es periódica de orden p si se trata de un $p - ciclo$.

1.1.1. Representación gráfica de una órbita.

La representación gráfica de una órbita consiste en colocar los enteros positivos en el eje de abscisas, y en el eje de ordenadas los distintos valores de x_k, $k = 0, 1, 2, ...$ Dado que los valores de x_k se representan en el eje de ordenadas, la gráfica de la órbita es la proyección, sobre el eje vertical, de la solución. Por lo tanto, la representación gráfica de la órbita se puede obtener a partir de la gráfica de la correspondiente solución.

Veamos en la Figura 1 la representación gráfica de la solución de la ecuación de Malthus (1.12).

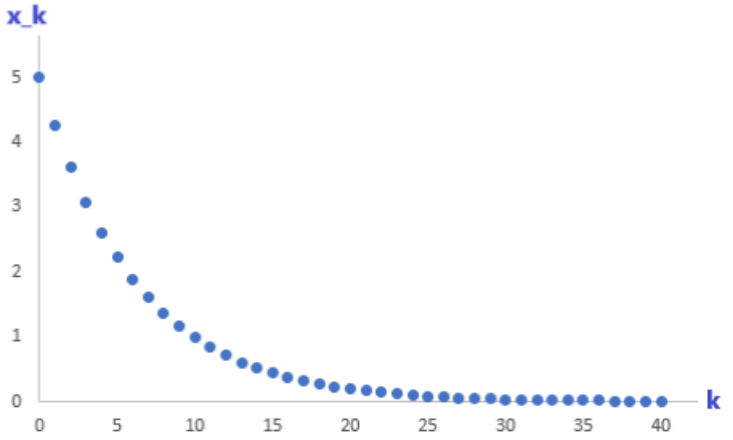

Figura 1: Representación de la solución de la ecuación de Malthus, $x_{k+1} = (1 + \alpha)x_k$, tomando α = -0.15 y $x_0 = 5$.

Como hemos mencionado anteriormente, en general no es posible determinar explícitamente las soluciones de una ecuación en diferencias, sobre todo cuando se trata de ecuaciones no lineales. Por ello, hay otro método para representar gráficamente un órbita que no requiere del cálculo de la solución. Este método es el llamado *diagrama de pasos* o *diagrama cobweb*.

El procedimiento para construir un diagrama de pasos es el siguiente:
En primer lugar se representan, en el plano xy, la recta $y = x$ y la gráfica de la función $y = f(x)$. Las soluciones constantes las determinan los puntos de intersección de la gráfica de la función f con la recta $y = x$. Para el resto de soluciones, dado un punto inicial x_0, el siguiente punto de la órbita es $x_1 = f(x_0)$. Podemos proyectar x_1, con la ayuda de la recta $y = x$, en el eje horizontal y aplicar nuevamente la función $f(x)$ para obtener x_2. Realizando reiteradamente este proceso obtenemos los sucesivos puntos que forman la órbita de la ecuación $x_{k+1} = f(x_k)$.

Vamos a representar en la Figura 2, con la ayuda de un programa específico para la construcción de diagramas de pasos [3], el *diagrama de pasos* para el modelo $x_{k+1} = \sqrt{x_k}+1$, tomando $x_0 = 0.25$.

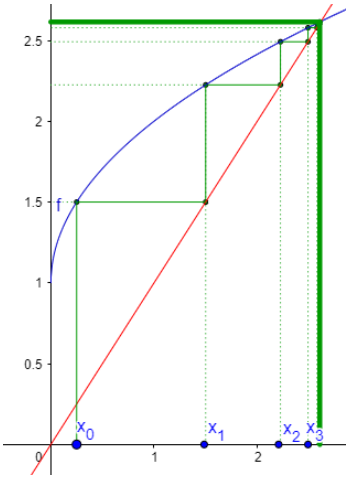

Figura 2: Diagrama de pasos de la ecuación no lineal $x_{k+1} = \sqrt{x_k} + 1$.

En la Figura 2, podemos apreciar que existe una única solución constante $x_k = c$, la cual se corresponde con el único punto de corte de la gráfica, representada en color azul, de $f(x) = \sqrt{x}+1$ con la recta $y = x$.

1.2. Estabilidad.

Otro concepto importante y que hay que tener en cuenta en el estudio de ecuaciones en diferencias es el de estabilidad. Como veremos ahora, según el comportamiento de las soluciones próximas a las soluciones constantes de la ecuación (1.1), éstas se podrán clasificar en: estables, asintóticamente estables o inestables.

Definición 1.2.1. Diremos que una solución constante $x_k \equiv c$ es *estable* si $\forall\ \epsilon > 0, \exists\ \delta > 0$ tal que si $|x_0 - c| < \delta \Longrightarrow |f^k(x_0) - c| = |x_k - c| < \epsilon$.

Definición 1.2.2. Diremos que una solución constante $x_k \equiv c$ es *asintóticamente estable* si es estable y además $\exists\ \delta > 0$ tal que si $|x_0 - c| < \delta \Longrightarrow \lim_{k\to\infty}(f^k(x_0)) = \lim_{k\to\infty} x_k = c$.

Definición 1.2.3. Diremos que una solución constante $x_k = c$ es *inestable* si no es estable, es decir, si $\exists\ \epsilon > 0$ tal que $\forall\ \delta > 0$ podemos encontrar x_0 con $|x_0 - c| < \delta$ y $|f^k(x_0) - c| = |x_k - c| > \epsilon$, para algún $k \geq 0$.

Más informalmente, diremos que un equilibrio es estable si las soluciones inicialmente próximas a él con el paso del tiempo siguen cerca de éste. Un equilibrio será asintóticamente estable si las órbitas que comienzan cerca, cuando $k \longrightarrow \infty$ tienden hacia él. Y, por el contrario, el equilibrio será inestable cuando existen puntos cercanos que se van alejando.

En ocasiones, el análisis gráfico es muy útil para averiguar la estabilidad de una solución constante. Los diagramas de pasos, como ya hemos mencionado antes, permiten analizar el comportamiento de las órbitas inicialmente cercanas a una solución constante $x_k \equiv c$ de la ecuación en diferencias. Por ejemplo, en la Figura 2 observamos que para cualquier punto inicial x_0, distinto del equilibrio, la trayectoria evoluciona acercándose al equilibrio cuando $k \longrightarrow \infty$, esto implica que es un atractor.

1.2.1. Criterios para el estudio de la estabilidad de una solución constante.

Una vez definidos estos conceptos y recordado el método gráfico del diagrama de pasos, vamos a mencionar un destacado criterio para determinar la estabilidad de una solución constante. Para enunciar y demostrar este teorema, se han tenido en cuenta las siguientes notas [2] y [17] :

Teorema 1.2.1.1. *Supongamos que $x_k \equiv c$ es una solución constante de la ecuación en diferencias $x_{k+1} = f(x_k)$ y que además, $f \in C^1(\mathbb{R})$. Entonces:*

- *Si $|f'(c)| < 1$, la solución constante $x_k \equiv c$ es asintóticamente estable.*
- *Si $|f'(c)| > 1$, la solución constante $x_k \equiv c$ es inestable.*

Demostración:

1. En primer lugar, veamos el caso $|f'(c)| < 1$:

   Por hipótesis, $f \in C^1 \Longrightarrow f'$ es continua , por lo que podemos afirmar que $\exists\ 0 < M < 1$ y $d > 0$ tal que $|f'(x)| \leq M < 1$, $\forall\ x \in (c-d, c+d)$.

   Consideremos $x_0 \in (c-d, c+d)$, por el Teorema del Valor Medio $\exists\ p \in (c-d, c+d)$ tal que:

   $$x_1 - c = f(x_0) - f(c) = f'(p)(x_0 - c).$$

   Tomando valores absolutos en dicha igualdad obtenemos:

   $$|x_1 - c| = |f^{'}(p)|\ |x_0 - c| \leq M|x_0 - c|,$$

   luego $|x_1 - c| \leq M|x_0 - c| \leq Md < d \Longrightarrow x_1 \in (c-d, c+d)$.

Análogamente, $\exists\ p \in [x_1, c] \subseteq (c-d, c+d)$ tal que:

$$|x_2 - c| = |f(x_1) - c| = |f'(p)||x_1 - c| \leq M|x_1 - c| \leq M^2|x_0 - c|$$

Repitiendo el procedimiento k veces, llegamos a la expresión:

$$|x_k - c| \leq M^k|x_0 - c| \to 0 \text{ cuando } k \to \infty, \text{ ya que por hipótesis } 0 < M < 1.$$

Por tanto, $x_k \to c$ cuando $k \to \infty$, con lo cual c es asintóticamente estable como se quería demostrar.

Veamos ahora el otro caso:

2. Si $|f'(c)| > 1$:

   De nuevo por continuidad de f', $\exists\ d > 0$ y $M > 1$ tales que $|f'(x)| > M$, $\forall\ x \in (c-d, c+d)$. Por tanto, aplicando de nuevo el Teorema del Valor Medio, $\exists\ p \in (c-d, c+d)$ tal que:

   $$|x_1 - c| = |f'(p)|\ |x_0 - c| > M|x_0 - c|$$

   Tras k pasos:

   $$|x_k - c| > M^k|x_0 - c|$$

   Como $M > 1$, tenemos que $M^k \to \infty$ cuando $k \to \infty$ y , por tanto, x_k se aleja de c, con lo que c es inestable.

■

Ejemplo 1.2.1.1. La ecuación $x_{k+1} = x_k^2$ tiene dos equilibrios, los cuáles corresponden a las dos soluciones de $x = x^2$, es decir, los equilibrios son $c_1 = 0$ y $c_2 = 1$. Observamos que:

1. $|f'(c_1)| = |f'(0)| = 0 < 1 \Longrightarrow c_1 = 0$ es asintóticamente estable.
2. $|f'(c_2)| = |f'(1)| = 2 > 1 \Longrightarrow c_2 = 1$ es inestable.

Si miramos la situación gráficamente, ver Figura 3, podemos deducir que para un dato inicial $x_0 \in (0, 1)$, las soluciones se alejan de $c_2 = 1$, el equilibrio que hemos comprobado que era inestable aplicando el teorema 1.2.1.1 y se acercan a $c_1 = 0$, el equilibrio asintóticamente estable .

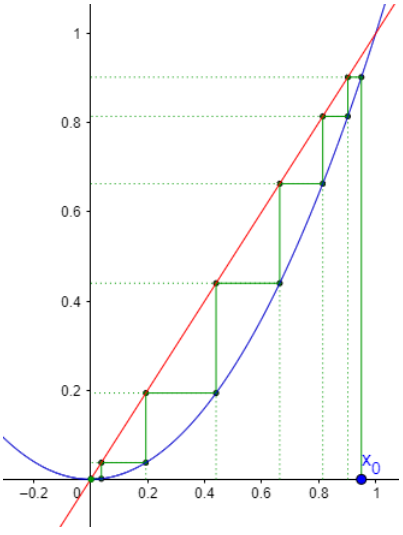

Figura 3: Diagrama de pasos de la ecuación $x_{k+1} = x_k^2$.

Definición 1.2.1.1. Dada una solución constante $x_k \equiv c$, en las condiciones del teorema 1.2.1.1, diremos que c es un *punto hiperbólico* si $|f'(c)| \neq 1$, *superatractivo* si $|f'(c)| = 0$ y *neutral* si $|f'(c)| = 1$.

Cabe destacar que el teorema 1.2.1.1 permite el estudio de la estabilidad o inestabilidad de la solución constante $x_k \equiv c$ en el caso en el c sea un punto hiperbólico. Por ello, cuando el teorema 1.2.1.1 no es concluyente es necesario un estudio más profundo.

Definición 1.2.1.2. Una solución constante $x_k \equiv c$ de la ecuación en diferencias $x_{k+1} = f(x_k)$ se dice que es:

- *semiestable desde arriba (abajo)* si $\forall\, \epsilon > 0$, $\exists\, \delta > 0$ tal que si $c \leq x_0 < c + \delta$ $(c - \delta < x_0 \leq c)$, entonces $|f^k(x_0) - c| = |x_k - c| < \epsilon$, $\forall k \in \mathbb{N}$.
- *atractiva desde arriba (abajo)* si $\exists\, \mu > 0$ tal que si $c \leq x_0 < c + \mu$ $(c - \mu < x_0 \leq c)$, entonces $\lim_{k\to\infty} f^k(x_0) = \lim_{k\to\infty} x_k = c$.
- *asintóticamente estable desde arriba (abajo)* si es semiestable desde arriba (abajo) y atractiva desde arriba (abajo).
- *inestable desde arriba (abajo)* si no es semiestable desde arriba (abajo).

Teorema 1.2.1.2. *Sea la ecuación en diferencias $x_{k+1} = f(x_k)$ donde $f \in C^1(\mathbb{R})$ y $x_k \equiv c$ es una solución constante con $f'(c) = 1$:*

- *Si f es convexa en un entorno de c, entonces $x_k \equiv c$ es semiestable desde abajo.*
- *Si f es cóncava en un entorno de c, entonces $x_k \equiv c$ es semiestable desde arriba.*

Además, cuando la convexidad (o concavidad) es estricta, $x_k \equiv c$ es asintóticamente estable desde abajo e inestable desde arriba (o asintóticamente estable desde arriba e inestable desde abajo).

Demostración:
En el primer caso, partimos de que f es convexa en un entorno de c, por lo que se cumple la siguiente desigualdad:

$$f(x) \geq f(c) + f'(c)(x - c) = f(c) + (x - c) = c + (x - c) = x, \tag{1.13}$$

por tanto $f(x) \geq x$. En consecuencia tenemos que $x_{k+1} = f(x_k) \geq x_k$.
Por otro lado, por la continuidad de f' podemos suponer que $f'(x) > 0$, $\forall x \in (c-\delta, c+\delta)$. Entonces, tomamos x_0 tal que $[x_0, c] \subseteq (c - \delta, c]$ y aplicando ahora el Teorema del Valor Medio: $\exists\, p \in (x_0, c)$ tal que:

$$f'(p) = \frac{f(c) - f(x_0)}{c - x_0} = \frac{c - x_1}{c - x_0}$$

y como $f'(x) > 0$, $\forall x \in (c - \delta, c + \delta)$, entonces $f'(p) > 0$, por lo que $x_1 < c$.

Una vez demostrado esto, reiterando el proceso anterior llegamos a que $\{x_k\}$ es una sucesión creciente y acotada superiormente por c.

Si tomamos ahora $x_0 \in (c, c + \delta)$ aplicando de nuevo la desigualdad (1.13) llegamos a que $x_1 \geq x_0 > c$. Por tanto, del mismo modo que anteriormente, llegamos a que $\{x_k\}$ es una sucesión creciente. Pero en este caso, a diferencia que en el anterior, $\{x_k\}$ se va alejando de c, luego es inestable por arriba.

En conclusión, si f es convexa en un entorno del punto fijo c tenemos que c es semiestable desde abajo, como se quería demostrar.

En el otro caso, partimos de que f es cóncava en un entorno de c, por tanto podemos concluir con un procedimiento análogo mediante la desigualdad:

$$f(x) \leq f(c) + f'(c)(x-c) = f(c) + (x-c) = x, \tag{1.14}$$

que para $x_0 \in (c, c+\delta)$, $\{x_k\}$ es una sucesión decreciente y acotada inferiormente por c.

Por otro lado, si $x_0 \in (c-\delta, c)$, aplicando la desigualdad (1.14) llegamos a que $x_1 \leq x_0 < c$. Entonces, de la misma manera, $\{x_k\}$ es una sucesión decreciente y además se aleja de c, por tanto es inestable desde arriba.

Finalmente, concluimos que si f es cóncava en un entorno de c entonces es semiestable desde arriba.

■

Veamos gráficamente mediante las siguientes figuras el significado del teorema:

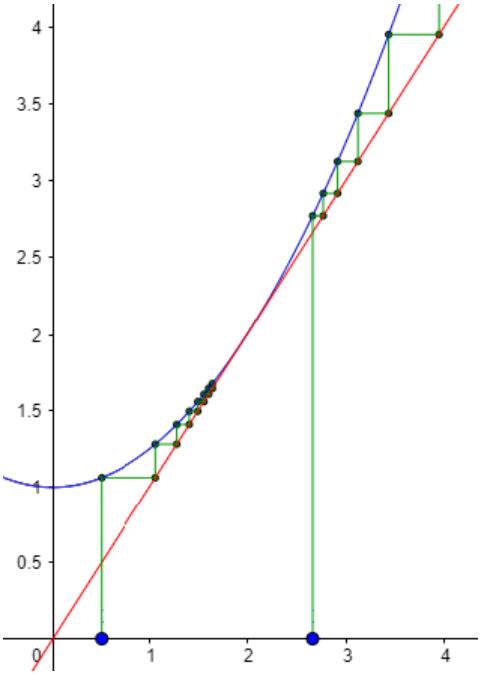

Figura 4: Diagrama de pasos para la ecuación en diferencias $x_{k+1} = f(x_k)$ siendo f una función convexa.

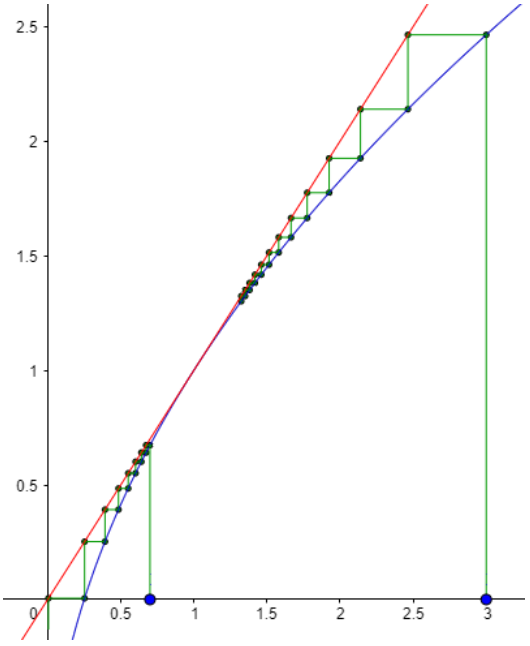

Figura 5: Diagrama de pasos para la ecuación en diferencias $x_{k+1} = f(x_k)$ siendo f una función cóncava.

En efecto, en la Figura 4 tenemos que f se trata de una función convexa y podemos observar como para un x_0 por debajo del punto de equilibrio c, las soluciones se van acercando hacia él. Sin embargo, para un valor inicial a la derecha de mi solución constante, las soluciones se alejan de ésta. Esto quiere decir que, efectivamente, la solución constante es asintóticamente estable desde abajo e inestable desde arriba.

Por el contrario, la Figura 5 nos muestra que cuando f es una función cóncava, para un valor inicial a la derecha de la solución constante c, la sucesión de las soluciones convergen hacia ella, mientras que si tomamos un valor a la izquierda de c se alejan de ésta.

Otra forma de ver el teorema anterior es la siguiente:

Teorema 1.2.1.3. *Sea la ecuación en diferencias $x_{k+1} = f(x_k)$ donde $f \in C^2(R)$ y $x_k \equiv c$ es una solución constante con $f'(c) = 1$:*

- *Si $f''(c) > 0$,entonces $x_k \equiv c$ es asintóticamente estable desde abajo y repulsivo desde arriba.*
- *Si $f''(c) < 0$, entonces $x_k \equiv c$ es asintóticamente estable desde arriba y repulsivo desde abajo.*

Cuando $f''(c) = 0$ podemos seguir con la prueba basada en la convexidad o concavidad de f en un entorno de la solución constante $x_k \equiv c$ llegando al siguiente resultado:

Teorema 1.2.1.4. *Supongamos que $x_k \equiv c$ es una solución constante de la ecuación en diferencias $x_{k+1} = f(x_k)$. Si además $f \in C^3(\mathbb{R})$, $f'(c) = 1$ y $f''(c) = 0$:*

- *Si $f'''(c) < 0$, entonces la solución constante $x_k \equiv c$ es asintóticamente estable.*
- *Si $f'''(c) > 0$, entonces la solución constante $x_k \equiv c$ es inestable.*

Demostración:

- Si partimos de que $f'''(c) < 0$, entonces f'' es decreciente en un entorno de c. Por tanto:

$$f''(x) > 0, \forall x \in (c - \delta, c)$$

$$f''(x) < 0, \forall x \in (c, c + \delta)$$

  La primera desigualdad nos asegura que $\forall x \in (c - \delta, c)$, f es estrictamente convexa en dicho intervalo y el teorema 1.2.1.2 nos dice que c es asintóticamente estable desde abajo, es decir, asintóticamente estable en el intervalo $(c - \delta, c)$.

  Por otro lado, la segunda desigualdad afirma que $\forall x \in (c, c+\delta)$, f es estrictamente cóncava y, aplicando de nuevo el teorema 1.2.1.2, tenemos que c es asintóticamente estable desde arriba.

  Por tanto, podemos afirmar que c es asintóticamente estable $\forall x \in (c - \delta, c + \delta)$.

- En el otro caso, si $f'''(c) > 0$, f'' es creciente en un entorno de c, y tendríamos:

$$f''(x) < 0, \forall x \in (c - \delta, c)$$

$$f''(x) > 0, \forall x \in (c, c + \delta)$$

  Nuevamente, la primera desigualdad nos asegura que $\forall x \in (c - \delta, c)$, f es cóncava en dicho intervalo y el teorema 1.2.1.2 nos afirma que c es semiestable desde arriba, luego en el intervalo $(c - \delta, c)$ es inestable.

  Mientras que, la segunda desigualdad nos dice que $\forall x \in (c, c + \delta)$, f es convexa y, aplicando de nuevo el teorema 1.2.1.2 tenemos que c es semiestable desde abajo y, de nuevo, eso implica que en el intervalo $(c, c + \delta)$ es inestable. En conclusión, c es inestable.

■

Lema 1.2.1.1. *Sea c una solución constante de $x_{k+1} = f(x_k)$. Entonces $x_k \equiv c$ es una solución constante asintóticamente estable para $x_{k+1} = f(x_k) \iff x_k \equiv c$ es una solución constante asintóticamente estable para $x_{k+1} = f^2(x_k)$.*

Demostración:
Ya sabemos que si c es un punto de equilibrio de f también lo es de f^2. Supongamos que c es asintóticamente estable para $x_{k+1} = f(x_k)$: $\forall \epsilon > 0$, $\exists\ \delta > 0$ tal que si $x_0 \in (c-\delta, c+\delta)$, entonces $f^k(x_0) = x_k \in (c-\epsilon, c+\epsilon)$, $\forall k$ y $\lim_{k\to\infty} f^k(x_0) = c$. Como estas relaciones se mantienen cuando k es par, tenemos que c es un punto de equilibrio asintóticamente estable para $x_{k+1} = f^2(x_k)$.

Para la otra implicación, asumimos ahora que c es una solución constante asintóticamente estable para $x_{k+1} = f^2(x_k)$. Tenemos entonces que $\forall \epsilon > 0$, $\exists\ \bar{\delta} > 0$ tal que si $\bar{x}_0 \in (c-\bar{\delta}, c+\bar{\delta})$, entonces $f^{2k}(\bar{x}_0) = \bar{x}_k \in (c-\epsilon, c+\epsilon)$, $\forall k$ y $\lim_{k\to\infty} f^{2k}(\bar{x}_0) = c$.
Faltaría examinar las iteraciones impares de f. Debido a la continuidad de f en c sabemos que $\exists\ \delta,\ 0 < \delta \leq \bar{\delta}$, tal que, si fijamos $\epsilon_1 = min(\epsilon, \delta)$ con $x_0 \in (c-\delta, c+\delta)$ podemos deducir que $f(x_0) \in (c-\epsilon_1, c+\epsilon_1)$. Entonces, $f^{2k+1}(x_0) = f^{2k}(f(x_0)) = f^{2k}(\bar{x}_0) \in (c-\epsilon, c+\epsilon)$ y $\lim_{k\to\infty} f^{2k+1}(x_0) = \lim_{k\to\infty} f^{2k}(f(x_0)) = f^{2k}(\bar{x}_0) = c$.

■

En el caso en el que $f'(c) = -1$ podemos aplicar el siguiente teorema para averiguar la estabilidad de la solución constante:

Teorema 1.2.1.5. *Supongamos que $x_k \equiv c$ es una solución constante de la ecuación en diferencias $x_{k+1} = f(x_k)$. Si además $f \in C^3(\mathbb{R})$ y $f'(c) = -1$:*

- *Si $2f'''(c) + 3(f''(c))^2 > 0$, entonces la solución constante $x_k \equiv c$ es asintóticamente estable.*
- *Si $2f'''(c) + 3(f''(c))^2 < 0$, entonces la solución constante $x_k \equiv c$ es inestable.*

Demostración:
En primer lugar, sabemos que si c es un punto fijo de f también lo es de f^2, ya que si llamamos $g(c) := f^2(c) = f(f(c)) = f(c) = c$, entonces $g(c) = c$. Además, si c es asintóticamente estable respecto de g ya vimos, por el lema 1.2.1.1, que también lo es respecto de f. Además:

$$g'(x) = \frac{\partial}{\partial_x} f(f(x)) = f'(f(x))f'(x),$$

de modo que:

$$g'(c) = (f(f(c)))' = f'(f(c))f'(c) = f'(c)f'(c) = 1.$$

Veamos ahora cuál es el valor de la segunda derivada:

$$g''(c) = (f'(f(c))f'(c))' = f''(f(c))(f'(c))^2 + f'(f(c))f''(c) = f''(c) - f''(c) = 0.$$

Por tanto, de acuerdo con el teorema 1.2.1.4 y el lema 1.2.1.1, el comportamiento de las órbitas dependerá del signo de g''':

$$g'''(c) = (f''(f(c))(f'(c))^2 + f'(f(c))f''(c))' =$$
$$f'''(f(c))(f'(c))^3 + 3f''(f(c))f''(c)f'(c) + f'(f(c))f'''(c) = -2f'''(c) - 3f''(c)^2$$

Distinguiendo los dos casos del teorema 1.2.1.5 tenemos que:

- Si $2f'''(c) + 3f''(c)^2 > 0$, entonces $g'''(c) = -(2f'''(c) + 3f''(c)^2) < 0$. El teorema 1.2.1.4 nos dice que c es asintóticamente estable.
- Si $2f'''(c) + 3f''(c)^2 < 0$, entonces $g'''(c) = -(2f'''(c) + 3f''(c)^2) > 0$. El teorema 1.2.1.4 nos dice que c es inestable.

Concluyéndose así la demostración.

■

Cabe destacar que la expresión $Sf(x) := \frac{f'''(x)}{f'(x)} - \frac{3}{2}(\frac{f''(x)}{f'(x)})^2$ es la conocidad *derivada schwarziana*. Teniendo en cuenta que, en las condiciones del teorema anterior, $f'(c) = -1$ tenemos que:

$$Sf(c) = -f'''(c) - \tfrac{3}{2}(f''(c))^2.$$

Por tanto, podemos relacionar la derivada schwarziana con $g'''(c)$ de la siguiente manera: $g'''(c) = 2Sf(c)$.

En consecuencia, podemos reescribir el teorema 1.2.1.5 de la siguiente forma:

Teorema 1.2.1.6. *Supongamos que $x_k \equiv c$ es una solución constante de la ecuación en diferencias $x_{k+1} = f(x_k)$. Si además $f \in C^3(\mathbb{R})$ y $f'(c) = -1$:*

- *Si $Sf(c) < 0$, entonces la solución constante $x_k \equiv c$ es asintóticamente estable.*
- *Si $Sf(c) > 0$, entonces la solución constante $x_k \equiv c$ es inestable.*

1.2.2. Estabilidad de los ciclos.

Una vez visto cómo se estudia la estabilidad de una solución constante, el estudio de la estabilidad de los ciclos resultará sencillo. Esto se debe a que, como ya explicamos, todo ciclo de orden p de la forma $\{c_0, c_1, ..., c_{p-1}, c_0, c_1, , ..., c_{p-1}, ...\}$, se corresponde con la solución constante del sistema $x_{k+1} = f^p(x_k)$, por lo que el análisis de la estabilidad del ciclo de f se puede llevar a cabo estudiando el de las soluciones constantes de f^p. Esta sección ha sido elaborada con la ayuda del capítulo 3 del libro [17].

Por simplicidad, empecemos considerando los $2-$*ciclos*. Recordemos que el par $\{c_0, c_1\}$ nos da un $2-$*ciclo* no trivial para la ecuación en diferencias $x_{k+1} = f(x_k)$ si:

$$c_0 \neq c_1, \quad f(c_0) = c_1, \quad f(c_1) = c_0,$$

concluyendo entonces que :

$$f^{2k}(c_0) = c_1 \quad \text{y} \quad f^{2k+1}(c_1) = c_0, \quad \forall k \geq 0.$$

Por otro lado, como hemos mencionado anteriormente, $\{c_0, c_1\}$ da un $2-$*ciclo* no trivial de la ecuación en diferencias $x_{k+1} = f(x_k) \iff c_0$ y c_1 son soluciones constantes de $x_{k+1} = f^2(x_k)$, pero no de $x_{k+1} = f(x_k)$.

Definición 1.2.2.1. Diremos que un $2-$*ciclo* de la ecuación $x_{k+1} = f(x_k)$ es estable (respectivamente, asintóticamente estable, inestable), si y sólo si, c_0 y c_1 son soluciones constantes estables (respectivamente, asintóticamente estables, inestables) de $x_{k+1} = f^2(x_k)$.

Aplicando ahora el teorema 1.2.1.1 a las soluciones constantes de la ecuación $x_{k+1} = f^2(x_k)$ llegamos al siguiente resultado:

Lema 1.2.2.1. Sea el $2-$*ciclo* generado por $\{c_0, c_1\}$, entonces:

- Si $|(f^2)'(c_0)| < 1$ y $|(f^2)'(c_1)| < 1$, entonces el $2-$*ciclo* generado por $\{c_0, c_1\}$ es asintóticamente estable.
- Si $|(f^2)'(c_0)| > 1$ o $|(f^2)'(c_1)| > 1$, entonces el $2-$*ciclo* generado por$\{c_0, c_1\}$ es inestable.

En ocasiones, comprobar las condiciones del lema anterior puede resultar laborioso. Una forma equivalente y, quizás más eficiente para estudiar la estabilidad de los $2-$*ciclos* es el siguiente teorema.

Teorema 1.2.2.1. *Sea $x_{k+1} = f(x_k)$ una ecuación en diferencias donde $f \in C^1(\mathbb{R})$ y $\{c_0, c_1\}$ genera un $2-ciclo$ de ésta.*

- *Si $|f'(c_0)f'(c_1)| < 1$, entonces el ciclo generado por $\{c_0, c_1\}$ es asintóticamente estable.*
- *Si $|f'(c_0)f'(c_1)| > 1$, entonces el ciclo generado por $\{c_0, c_1\}$ es inestable.*

Demostración:

$$(f^2)'(c_0) = f'(f(c_0))f'(c_0) = f'(c_1)f'(c_0) = f'(c_1)f'(f(c_1)) = (f^2)'(c_1)$$

Aplicando ahora el lema 1.2.2.1:

- Si $|(f^2)'(c_0)| = |(f^2)'(c_1)| = |f'(c_0)f'(c_1)| < 1 \Longrightarrow$ el $2-ciclo$ es asintóticamente estable.
- Si $|(f^2)'(c_0)| = |(f^2)'(c_1)| = |f'(c_0)f'(c_1)| > 1 \Longrightarrow$ el $2-ciclo$ es inestable.

■

Tras este teorema, la pregunta que nos debería de surgir es si es posible enunciarlo para $p-ciclos$ en general. Observemos que si $\{c_0, c_1, ..., c_{p-1}\}$ genera un $p-ciclo$ de la ecuación en diferencias $x_{k+1} = f(x_k)$ donde $f \in C^1(\mathbb{R})$, entonces para $i = 0, 1, 2, ..., p-1$ se cumple que:

$$(f^p)'(c_i) = f'(f^{p-1}(c_i))f'(f^{p-2}(c_i))...f'(c_i) = f'(c_0)f'(c_1)...f'(c_{p-1})$$

Por tanto, aplicando el mismo argumento que el utilizado en la demostración del teorema 1.2.2.1, obtenemos la respuesta a la pregunta.

En efecto, una caracterización para el estudio de la estabilidad de los $p-ciclos$ nos la dará el siguiente teorema:

Teorema 1.2.2.2. *Sea $x_{k+1} = f(x_k)$ una ecuación en diferenicas donde $f \in C^1$ y $\{c_0, c_1 ..., c_{p-1}\}$ es un $p-ciclo$ de ésta:*

- *Si $|f'(c_0)f'(c_1)...f'(c_{p-1})| < 1$, entonces el $p-ciclo$ es asintóticamente estable.*
- *Si $|f'(c_0)f'(c_1)...f'(c_{p-1})| > 1$, entonces el $p-ciclo$ es inestable.*

1.3. Bifurcaciones en ecuaciones paramétricas.

Para comenzar este apartado, vamos a considerar una ecuación en diferencias de la siguiente forma:

$$x_{k+1} = f(a, x_k) \tag{1.15}$$

donde $a \in \mathbb{R}$ y f es una función continua de $\mathbb{R}^2$. Estas ecuaciones son llamadas ecuaciones paramétricas ya que dependen del parámetro a.

Vamos a ver, gracias a la ayuda de [10] y [14], cómo pequeños cambios en el valor del parámetro a pueden modificar por completo el comportamiento de las soluciones de la ecuación (1.15).

Definición 1.3.1. Un punto $\bar{a}$ se dirá que es un *punto de bifurcación* si se produce un cambio cualitativo de las soluciones para $a < \bar{a}$ y $a > \bar{a}$.

Definición 1.3.2. Llamaremos *diagrama de bifurcación* a la representación, en el plano a, x, de las curvas de las soluciones constantes, junto con la indicación de su estabilidad.

Para diferenciar unas de otras, a las curvas de las soluciones constantes asintóticamente estables las representaremos con un trazo continuo. Mientras que, a las de las soluciones constantes inestables con un trazo discontinuo.

Ejemplo 1.3.1. Si consideramos la ecuación:

$$x_{k+1} = a - x_k^2, \tag{1.16}$$

podemos calcular sus soluciones constantes resolviendo, como ya hemos visto, la ecuación:

$$x = a - x^2 \quad \text{o, equivalentemente,} \quad x^2 + x - a = 0,$$

obteniendo como resultado: $c_1 = \frac{-1+\sqrt{1+4a}}{2}$ y $c_2 = \frac{-1-\sqrt{1+4a}}{2}$.

Es obvio, que para que c_1 y c_2 estén en $\mathbb{R}$ debe cumplirse que $a \geq \frac{-1}{4}$. Concretamente, en el caso en el que $a = \frac{-1}{4}$ obtenemos que $c_1 = c_2 = \frac{-1}{2}$. El valor $a = \frac{-1}{4}$ es un punto de bifurcación.

Estudiemos ahora la estabilidad de las soluciones constantes c_1, c_2 mediante el teorema 1.2.1.1. En efecto, aplicando dicho teorema, llegamos a que:

$$f'(x) = -2x \Longrightarrow f'(c_1) = 1 - \sqrt{1+4a} < 1 \text{ y } f'(c_2) = 1 + \sqrt{1+4a} > 1$$

Por tanto, c_1 será asintóticamente estable cuando $1 - \sqrt{1+4a} > -1 \iff a < \frac{3}{4}$, ya que necesitamos que $|f'(c_1)| < 1$.Luego tenemos que $a \in (\frac{-1}{4}, \frac{3}{4})$. Por su parte, c_2 es siempre inestable para $a > \frac{-1}{4}$. Cabe destacar que otro punto de bifurcación es $a = \frac{3}{4}$. En tal caso, $c_1 = \frac{1}{2}$ y $c_2 = \frac{-3}{2}$.

Representando el valor de a en el eje de abscisas y el de x en el de ordenadas, mostremos en la Figura 6 el diagrama de bifurcación de la ecuación (1.16) :

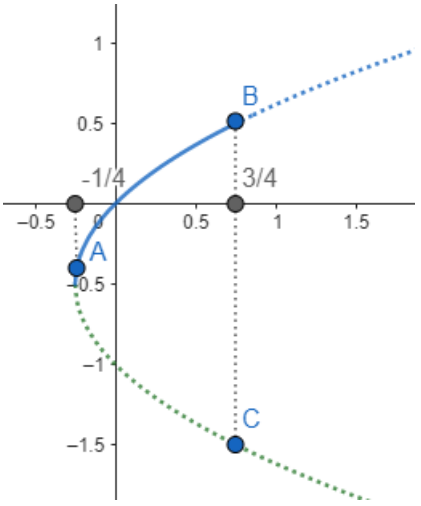

Figura 6: Diagrama de bifurcación de la ecuación $x_{k+1} = a - x_k^2$.

2. Capítulo 2: Ecuación logística discreta.

Este segundo capítulo está destinado a estudiar la ecuación logística discreta, que es la base de nuestro trabajo. En primer lugar, con la ayuda de los apuntes [11], presentaremos dicha ecuación para posteriormente estudiar algunos conceptos definidos en el capítulo anterior como son las soluciones constantes, los ciclos y la estabilidad. Finalmente, concluiremos el capítulo hablando del caos y todo lo que éste conlleva.

La ecuación logística discreta surge de la necesidad de corregir el modelo de Malthus. Recordemos, que el modelo de Malthus (1.12) sufría algunas imperfecciones, ya que suponía la tasa de crecimiento constante. Por tanto, para llegar a la ecuación que estamos buscando y evitar así la extinción o el crecimiento ilimitado, la solución será tomar una tasa de crecimiento no constante. Para ello, vamos a considerar una recta decreciente como función de p_n de esta forma:

$$\frac{p_{k+1}}{p_k} = a - bp_k, \quad a, b > 0.$$

Ahora, despejando p_k al término derecho llegamos a la ecuación logística:

$$p_{k+1} = (a - bp_k)p_k \iff p_{k+1} = ap_k(1 - \frac{bp_k}{a})$$

Al hacer el cambio de variable $x_k = \frac{bp_k}{a}$, obtenemos otra notación más común para esta ecuación, que es la siguiente:

$$x_{k+1} = \frac{bp_{k+1}}{a} = \frac{b}{a}(ap_k(1 - \frac{bp_k}{a})) = a(1 - \frac{bp_k}{a})\frac{bp_k}{a} = a(1 - x_k)x_k \Rightarrow x_{k+1} = a(1 - x_k)x_k$$

En conclusión, llegamos a que la forma general de la ecuación logística discreta es:

$$x_{k+1} = a(1 - x_k)x_k, \quad a > 0. \tag{2.1}$$

Una vez llegados a este punto, nos debería surgir la pregunta de qué condiciones debe cumplir dicha ecuación para que tenga sentido biológico. Para ello, en primer lugar necesitamos garantizar la no negatividad de x_{k+1}, pues no tendría sentido hablar de poblaciones negativas.

Observamos entonces que se debe cumplir que: $x_k \geq 0$ y que $1 - x_k \geq 0 \iff 0 \leq x_k \leq 1$.

Queremos, por tanto, que si $0 \leq x_0 \leq 1$, entonces también se deberá verificar que $0 \leq f(x_k) \leq 1$, $k = 0, 1, 2, \ldots$. Cabe observar que $f(x)$ es una parábola que corta al eje de abscisas en los puntos 0 y 1. Además, tenemos que el punto crítico de la función f es $(\frac{1}{2}, \frac{a}{4})$, puesto que:

$$f'(x) = a(1 - 2x) = 0 \iff x = \tfrac{1}{2} \text{ y } f(\tfrac{1}{2}) = \tfrac{a}{4}.$$

Dicho punto crítico, $(\frac{1}{2}, \frac{a}{4})$, se trata de un máximo.

Como necesitamos que $0 \leq f(x) \leq 1$, $x \in [0, 1] \Longrightarrow 0 \leq \frac{a}{4} \leq 1 \iff 0 \leq a \leq 4$.

Por tanto, concluímos que para que la ecuación logística discreta (2.1) tenga sentido biológico debe cumplir estas dos condiciones:

1. $0 \leq x_0 \leq 1$.
2. $0 \leq a \leq 4$.

Veamos gráficamente cómo varían los valores de la función de la ecuación logística discreta (2.1) dependiendo del valor del parámetro a.

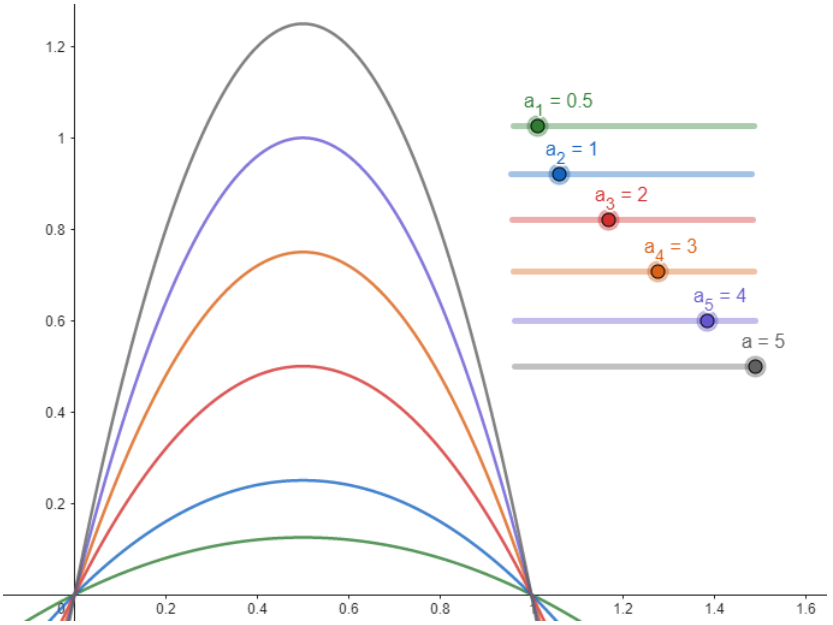

Figura 7: Función $f(x) = a(1-x)x$ para distintos valores de a.

En la Figura 7 observamos que para un valor de $a > 4$ la gráfica de f supera el valor 1.

2.1. Soluciones constantes.

Nuestro siguiente objetivo es encontrar las soluciones constantes de la ecuación logística discreta. En el capítulo anterior ya definimos lo que eran las soluciones constantes y la forma de hallarlas. En efecto, determinar las soluciones constantes de la ecuación logística equivale a resolver la ecuación $x = a(1-x)x$, obteniendo como soluciones: $c_1 = 0$ y $c_2 = 1 - \frac{1}{a}$.

En consecuencia, las soluciones constantes que buscábamos son:

- $x_k \equiv 0$.
- $x_k \equiv 1 - \frac{1}{a}$. En este caso, hay que tener en cuenta que la solución tiene sentido biológico sólo si $a > 1$.

Luego, concluimos que:

- Si $0 < a \leq 1 \Longrightarrow$ hay una única solución constante $x_k \equiv 0$.
- Si $0 < a \leq 4 \Longrightarrow$ hay dos soluciones constantes $x_k \equiv 0$ y $x_{\equiv} 1 - \frac{1}{a}$.

2.1.1. Estabilidad.

Tras el hallazgo de las soluciones constantes, la siguiente cuestión que nos surge es cuál será la estabilidad o inestabilidad de éstas. Para dar respuesta a esta pregunta, estudiaremos su estabilidad aplicando el Teorema 1.2.1.1:

En primer lugar, debemos calcular la derivada de $f(x) = a(1-x)x$, para posteriormente evaluarla en las soluciones constantes $c_1 = 0$ y $c_2 = 1 - \frac{1}{a}$:

$$f'(x) = a(1-2x).$$

En el caso en el que $c_1 = 0$, tenemos que $f'(0) = a$, por lo que debemos hacer una distinción de casos:

- Si $0 < a < 1 \Longrightarrow |f'(0)| = |a| < 1$. Aplicando ahora el teorema 1.2.1.1 tenemos que $x_k \equiv 0$ es asintóticamente estable.
- Si $a > 1 \Longrightarrow |f'(0)| = |a| > 1$. De igual forma, por el teorema 1.2.1.1 $x_k \equiv 0$ es inestable.

En efecto, las Figuras 8 y 9 muestran gráficamente el comportamiento del equilibrio $x_k \equiv 0$, dependiendo del valor que tome el parámetro a:

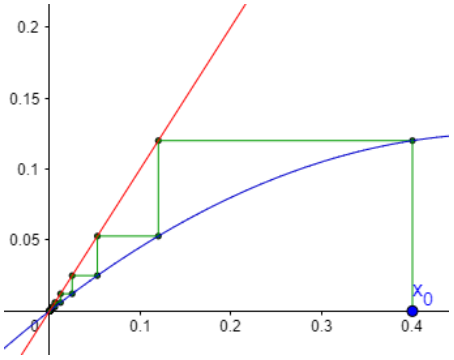

Figura 8: Diagrama de pasos para la ecuación logística con $a = 0.5$.

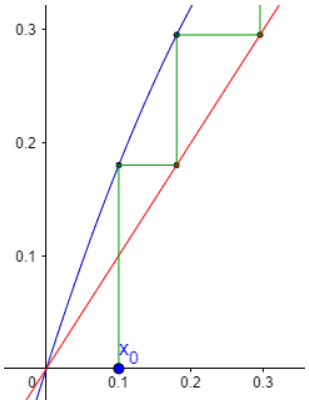

Figura 9: Diagrama de pasos para la ecuación logística con $a = 2$.

Por otro lado, para $c_2 = 1 - \frac{1}{a}$, obtenemos que $f'(1 - \frac{1}{a}) = a(1 - 2(1 - \frac{1}{a})) = 2 - a$. Ahora, haciendo de nuevo una distinción de casos, tenemos que:

- Si $a < 1$ no tiene sentido biológico, pero sí que lo tiene matemáticamente. Entonces, en tal caso, $|f'(1 - \frac{1}{a})| = |2 - a| > 1$ y, el Teorema 1.2.1.1 nos dice que $x_k \equiv 1 - \frac{1}{a}$ es inestable.
- Si $1 < a < 3 \Longrightarrow |f'(1 - \frac{1}{a})| = |2 - a| < 1$. Por el Teorema 1.2.1.1 tenemos que $x_k \equiv 1 - \frac{1}{a}$ es asintóticamente estable.
- Si $a > 3 \Longrightarrow |f'(1 - \frac{1}{a})| = |2 - a| > 1$. Nuevamente, por lo que el Teorema 1.2.1.1, $x_k \equiv 1 - \frac{1}{a}$ es inestable.

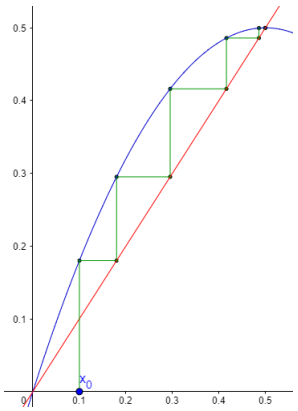

Figura 10: Diagrama de pasos para la ecuación logística con $a = 1.5$.

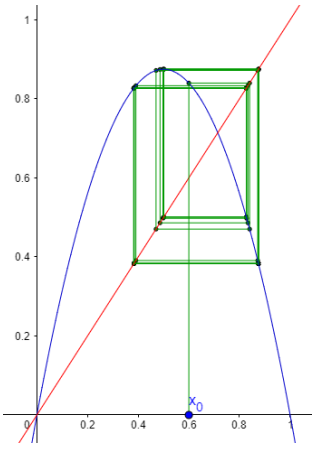

Figura 11: Diagrama de pasos para la ecuación logística con $a = 3.5$.

Observamos que los casos $a = 1$ y $a = 3$ quedan abiertos por el teorema 1.2.1.1, ya que tenemos que cuando $a = 1$ ambas soluciones constantes coinciden y $|f'(c_1)| = 1$. Por otro lado, cuando $a = 3$ tenemos los equilibrios $c_1 = 0$ y $c_2 = \frac{2}{3}$. Para el equilibrio $c_1 = 0$ el Teorema 1.2.1.1 sí nos da información mientras que para el equilibrio $c_2 = \frac{2}{3}$ no, puesto que $|f'(c_2)| = 1$ y ese caso queda abierto. Ambos puntos, tanto $a = 1$ como $a = 3$, son puntos de bifurcación.

Para el caso $a = 1$, aplicamos entonces el teorema 1.2.1.3:

Como $f''(c_1) = -2 < 0$, el teorema mencionado nos dice que $x_k \equiv 0$ es asintóticamente estable por arriba e inestable por abajo. Esto quiere decir que para cualquier condición inicial $x_0 > 0$ el equilibrio x_k es asintóticamente estable y que si $x_0 < 0$, el equilibrio es inestable. Pero recordamos que una de nuestras condiciones al desarrollar la ecuación logística era que $0 \leq x_0 \leq 1$, por lo que $x_k \equiv 0$ es asintóticamente estable para la ecuación $x_{k+1} = (1 - x_k)x_k$.

En el caso en el que $a = 3$, debemos aplicar el teorema 1.2.1.5:

En primer lugar, tenemos que $f''(c_2) = -6$ y que $f'''(c_2) = 0$. Aplicando ahora la fórmula del teorema:

$$2f'''(c_2) + 3(f''(c_2))^2 = 108 > 0$$

De lo que deducimos que para $a = 3$, el equilibrio $x_k \equiv 1 - \frac{1}{a} = \frac{2}{3}$ es asintóticamente estable.

Resumiendo el estudio de la estabilidad para las soluciones constantes $x_k \equiv 0$ y $x_k \equiv 1 - \frac{1}{a}$ tenemos que:

$0 < a \leq 1$	$c = 0$ es asintóticamente estable
$1 < a \leq 3$	$c = 0$ es inestable y $c = 1 - \frac{1}{a}$ asintóticamente estable
$3 < a \leq 4$	$c = 0$ y $c = 1 - \frac{1}{a}$ son inestables

2.2. Ciclos.

En el capítulo 1 destacamos dos tipos de soluciones particulares: las soluciones constantes y los ciclos. En el apartado anterior hemos visto cuáles son las soluciones constantes de la ecuación logística discreta y hemos estudiado su estabilidad para los distintos valores del parámetro a. En esta sección, con la ayuda de [14], haremos lo mismo pero con los ciclos. Para ello, lo primero que debemos hacer es estudiar para qué valores de a aparecen $p-$ciclos.

Comencemos en primer lugar con los $2 - ciclos$, que son los más sencillo:

Ya mencionamos que los $2-ciclos$ se determinan con las soluciones constantes de f^2: $x_0 = f^2(x_0)$. Si desarrollamos dicha expresión tenemos que:

$$x_0 = f(f(x_0)) = f(a(1 - x_0)x_0) = a(1 - a(1 - x_0)x_0)a(1 - x_0)x_0,$$

despejando todo al término izquierdo y sacando factor común x_0 llegamos a la siguiente ecuación de orden 4:

$$x_0(1 - a^2(a(1 - x_0)x_0)(1 - x_0)) = 0.$$

Por otro lado, si tenemos en cuenta que las soluciones constantes son, en particular, $2 - ciclos$, podemos dividir el polinomio $(1 - a^2(a(1 - x_0)x_0)(1 - x_0))$ por $x_0 - (1 - \frac{1}{a})$, obteniendo así:

$$x_0(ax_0 - (a - 1))(a^2x_0^2 - a(a + 1)x_0 + (a + 1)) = 0. \tag{2.2}$$

Por tanto, encontrar las soluciones de la ecuación (2.2) equivale a resolver las ecuaciones:

$$x_0 = 0, \quad ax_0 - (a-1) = 0 \quad \text{y} \quad a^2x_0^2 - a(a+1)x_0 + (a+1) = 0$$

Luego, los datos iniciales x_0 que necesitamos para que la solución sea un $2-ciclo$ son:

Por un lado, evidentemente, las soluciones constantes $x_0 = 0$ y $x_0 = 1 - \frac{1}{a}$. Por otro lado, las soluciones de la ecuación $a^2x_0^2 - a(a+1)x_0 + (a+1) = 0$, es decir, $x_0 = \frac{(a+1)+\sqrt{(a+1)(a-3)}}{2a}$ y $x_0 = \frac{(a+1)-\sqrt{(a+1)(a-3)}}{2a}$. Hay que tener en cuenta que para que estas dos últimas soluciones tengan sentido real el discriminante no puede ser negativo. Por tanto, solo existen $2-ciclos$ distintos de las soluciones constantes cuando $a > 3$.

Efectivamente, podemos observar en la Figura 12, como para un valor de $a < 3$ solo tenemos 2 soluciones constantes tanto en la primera como en la segunda iteración. Mientras que, para un valor de $a > 3$, la Figura 13 nos muestra que existen 4 soluciones constantes para la segunda iteración que se corresponden con los datos iniciales calculados anteriormente para que la ecuación tuviera $2-ciclos$.

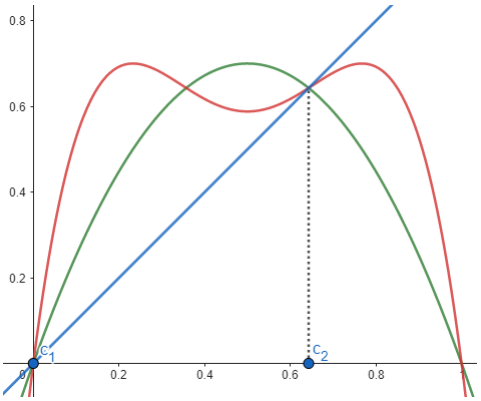

Figura 12: Primera (verde) y segunda (roja) iteración de la ecuación logística para $a = 2.8$.

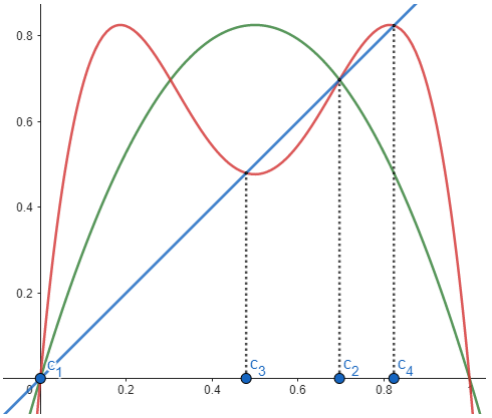

Figura 13: Primera (verde) y segunda (roja) iteración de la ecuación logística para $a = 3.3$.

En cuanto a la estabilidad de los $2-ciclos$, ya vimos en el anterior capítulo cómo se estudiaba. Si llamamos :

$$g(x) = f(f(x)) = a^2x(1-x)(1-ax(1-x)),$$

tras varias cuentas tenemos que

$$g'(x) = a^2(1-2x)(1-2ax(1-x)).$$

Si evaluamos g' en los puntos c_3 y c_4 ilustrados en la Figura 13, obtenemos que $g'(c_3) = g'(c_4) = -a^2 + 2a + 4$. Por tanto, al aplicar el Teorema 1.2.2.1, el $2-ciclo$ generado por $\{c_3, c_4\}$ será asintóticamente estable cuando $|-a^2 + 2a + 4| < 1$, es decir, si $3 < a < 1 + \sqrt{6}$.

A medida que a sigue aumentando los $2-ciclos$ se vuelven inestables. Por tanto, cada vez que a pasa por una serie de valores de bifurcación, el carácter de la solución va cambiando bruscamente. Esta situación la podemos ver reflejada en la Figura 18. A este tipo de bifurcación se le llama *bifurcación de duplicación de periodo*. En este tipo de bifurcaciones, la curva que surge está formada por puntos que generan los $2^n - ciclos$.

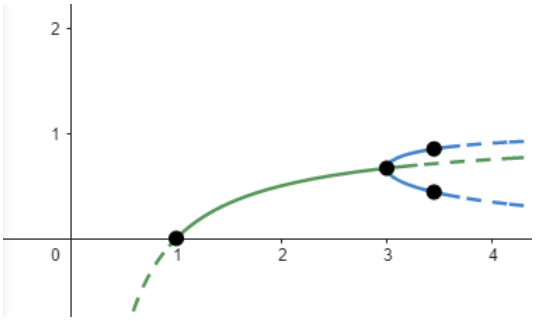

Figura 14: Diagrama de bifurcación de $2-ciclos$.

Si analizamos ahora la segunda iteración de g, es decir, la cuarta iteración de f, nos encontraremos con los $4-ciclos$. Plasmando de nuevo gráficamente las funciones f, f^2, f^4, observamos en la Figura 15, como cabía de esperar, que la cuarta iteración tiene cuatro jorobas y aparece una nueva solución periódica, un $4-ciclos$ generado por $\{c_5, c_6, c_7, c_8\}$.

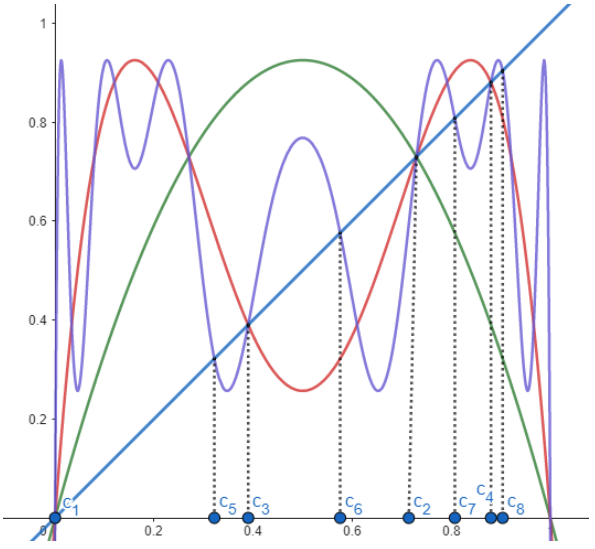

Figura 15: Primera (verde), segunda (roja) y cuarta (morada) iteración de la ecuación logística para $a = 3.7$.

Se ha podido demostrar [9] que conforme a va aumentando a través de las sucesivas bifurcaciones, cada solución par $p-$periódica se bifurca en una solución $2p-$periódica, y esto ocurre cuando a toma un valor tal que $|f'(c)| = 1$, siendo c la solución que da lugar a un $p-$ciclo, es decir, c se trata de un punto fijo de $f^p(x)$. Además, cabe destacar, que la distancia entre las bifurcaciones se va haciendo cada vez más pequeña.

Va a existir entonces, una jerarquía de 2^n-ciclos y, asociada a cada una un intervalo en el que el 2^n-ciclo es estable. Por ejemplo:

- Como ya hemos visto, si $3 < a < 1 + \sqrt{6}$, aparecen $2-ciclos$ y son estables.
- Si $1 + \sqrt{6} < a < 3.5$, aparecen $4-ciclos$ estables y los $2-ciclos$ se vuelven inestables.

Por otro lado, se puede demostrar, que existe un valor límite a_c en el que se produce la inestabilidad para todos los ciclos de orden 2^n. Es decir, para $a > a_c \approx 3.57$ todos los $2^n - ciclos$ se vuelven inestables. Se trata de un comportamiento bastante complejo y que se sale de los objetivos de este trabajo. Para $a > a_c$ comienzan a aparecer los ciclos impares, entre ellos los $3-ciclos$. Este tipo de ciclos se trata de un caso de estudio que veremos más detalladamente en la siguiente sección.

2.3. Caos.

Como hemos dicho anteriormente, cuando en la ecuación logística discreta se dan las condiciones para la aparición de un 3−ciclo, éste se trata de un caso de estudio particular. Para elaborar esta sección se han seguido las referencias [10] y [14] .

La Figura 16 nos muestra el comportamiento de la ecuación logística discreta a medida que a aumenta.

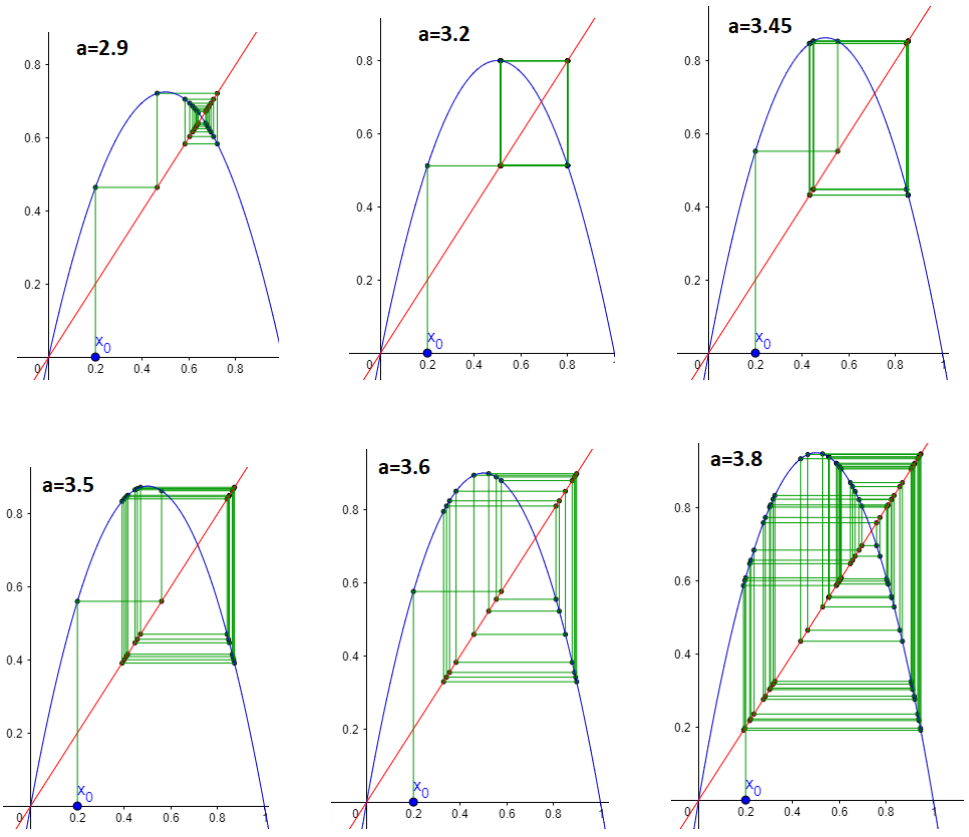

Figura 16: Evolución de las órbitas de la ecuación logística dependiendo del valor del parámetro a. Todas estas gráficas han sido realizadas mediante [3], un programa de geogebra específico para la construcción de diagramas de pasos.

En efecto, como ya demostramos, para un valor de a tal que $1 < a < 3$, las órbitas convergen a la solución constante no nula. Por otro lado, cuando $3 < a < 1 + \sqrt{6}$ aparece un $2 - ciclo$ asintóticamente estable. En el caso en el que $a = 3{,}45$ aparece un $4 - ciclo$ asintóticamente estable y para $a = 3{,}5$ un $8 - ciclo$. Finalmente, para $a > a_c \approx 3.57$ los ciclos pares se vuelven inestables.

Cabe destacar, que las órbitas aperiódicas que aparacen para $a_c < a \leq 4$ muestran dependencia sensible respecto de los datos iniciales. Esto quiere decir que errores iniciales pequeños desembocan en grandes errores futuros. A este tipo de órbitas se les denomina *órbitas caóticas*. Pues bien, gracias al *teorema de Li-Yorke (1975) [12]* que afirma que la presencia de $3 - ciclos$ implica que existen $p-ciclos$ para cualquier p, se puede demostrar que para $a > a_c$ la ecuación logística discreta presenta órbitas caóticas.

Por otro lado, se deduce que el valor que debe tomar a para que aparezca un $3-ciclo$ es aproximadamente 3.828. Podemos ver reflejado en la Figura 17 como para $a = 3.828$ aparecen 3 nuevas soluciones constantes c_3, c_4, c_5 de la tercera interacción de f que formará el $3-ciclo$ $\{c_3, c_4, c_5\}$.

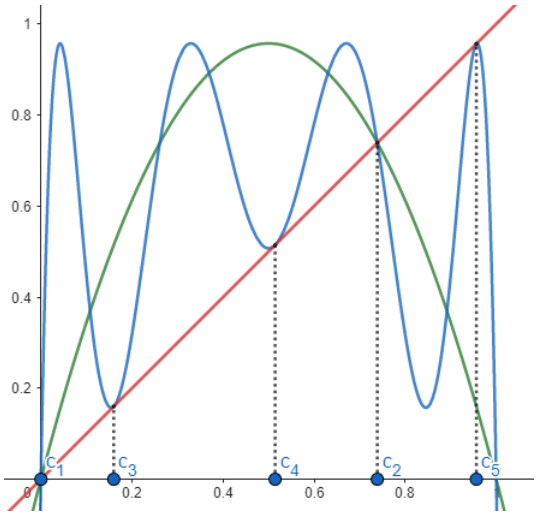

Figura 17: Primera (verde) y tercera (azul) iteración de f para $a = 3.828$.

Finalmente, concluiremos esta sección analizando el diagrama de bifurcación de la ecuación logística discreta que se muestra en la Figura 18.

En este diagrama vemos reflejado lo estudiado a lo largo de este capítulo. En efecto:

- Si $a < 1$ el atractor será la solución constante $c = 0$.
- Si $1 < a < 3$ el atractor será la solución constante $c = 1 - \frac{1}{a}$.
- Para $a = 3$ comienza lo que hemos llamado bifurcación de duplicación del periodo, apareciendo dos puntos periódicos estables que pierden su estabilidad en $a = 1 + \sqrt{6}$. Es en dicho punto donde aparece el $4-ciclo$.
- Para valores de a entre $a_c \approx 3.57$ y 4 las soluciones son caóticas.
- Para $a \approx 3.828$ aparece el $3-ciclo$.

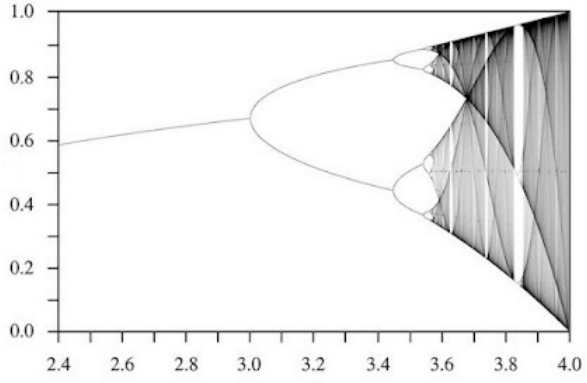

Figura 18: Diagrama de bifurcación, tomado de [4], de la ecuación logística discreta para valores de a en el intervalo [2.4,4].

3. Capítulo 3: Aplicación en criptografía.

La criptografía es un campo de estudio que tiene como objetivo garantizar la protección de la información en los entornos digitales. Hoy en día, debido al constante avance y al incremento del uso de las nuevas tecnologías, la seguridad en la comunicación y en el intercambio de datos se ha vuelto fundamental.

En estos últimos años, la rama de la criptografía basada en los sistemas caóticos ha ido ganando popularidad. Es aquí donde entra en juego lo estudiado a lo largo del trabajo, la ecuación logística discreta, pues su comportamiento caótico y altamente sensible a las condiciones iniciales ha dado lugar a su aplicación en esta rama. Esto se debe a que esta ecuación se puede utilizar para generar secuencias pseudoaleatorias. Estas secuencias generadas por la ecuación logística discreta son altamente impredecibles gracias a su naturaleza caótica lo que las hace adecuadas para aplicaciones de cifrado y generación de claves.

En este último capítulo, nuestro objetivo será la encriptación y desencriptación de una imagen haciendo uso del sistema caótico logístico discreto. Para ello, hemos seguido la idea explicada en [13] con la ayuda de [5]. Además, concluiremos el capítulo con algunas aportaciones que disminuirá notablemente el tiempo de ejecución del programa ideado por [13].

3.1. Esquema criptográfico.

En esta primera sección explicaremos, siguiendo el esquema de [13], cómo vamos a generar las secuencias pseudoaleatorias y el algoritmo de cifrado mediante la ecuación logística discreta.

En efecto, partimos de la ecuación logística discreta que, como ya explicamos en el capítulo anterior, recordamos que tiene la siguiente forma:

$$x_{k+1} = a(1 - x_k)x_k, \quad 0 \leq x_0 \leq 1 \ , \ 0 \leq a \leq 4$$

Utilizaremos la ecuación logística discreta como parte de un algoritmo de cifrado. El algoritmo tomará la información original de longitud L y la transformará en información cifrada de longitud LM.

El proceso de codificación explicado en [13] se basa en la generación de tres órbitas caóticas utilizando diferentes condiciones iniciales x_{0i} y parámetros a_i, $i = 1, 2, 3$, que actuarán como claves de cifrado y que serán utilizadas en diferentes etapas del proceso de cifrado. Hay que tener en cuenta que como las 3 órbitas que queremos generar deben ser caóticas se debe cumplir que $x_k \in (0, 1)$ y $a \in$ [3.57,4]. En la primera etapa del proceso, emplearemos la primera órbita para aplicar la técnica de difusión, es decir, se reorganizan los elementos de la información original asegurándonos así de que quede difusa y sea más difícil de descifrar en caso de atacantes. En la segunda etapa, se combina la segunda órbita con la información mezclada con el fin de aumentar su longitud de L a LM. Por último, la tercera órbita se utiliza para aplicar la técnica de confusión, que oculta la relación entre la información original, la información cifrada y las claves utilizadas, aumentando así aún más la seguridad del algoritmo.

3.2. Cifrado.

En este caso práctico abordamos la encriptación de una imagen, pero cabe destacar que estos algoritmos también podrían ser utilizados para codificar cualquier tipo de información.

Antes de explicar el proceso comentaremos cómo se entiende una imagen. Una imagen se trata de una matriz de números que representan la intensidad del color de los píxeles. Cada píxel se compone de tres bytes o subpíxeles, los cuales corresponden a los colores rojo (R), verde (G) y azul (B). Además, cada uno de estos subpíxeles es representado mediante un valor decimal que varía entre 0 y 255.

Para poder aplicar un algoritmo de cifrado a una imagen, utilizamos un vector que contenga los valores de los subpíxeles de ésta de la siguiente forma: $\{P_1^R, P_1^G, P_1^B, ..., P_n^R, P_n^G, P_n^B\}$. Tras almacenar la información en dicho vector, éste podrá ser manipulado para la realización del proceso de cifrado.

A continuación, vamos a explicar detalladamente el proceso que se sigue en [13] para la encriptación de una imagen mediante su implementación en Python.

En primer lugar, la notación que seguiremos a partir de ahora será la siguiente:

- L= longitud del vector con la información original.
- LM= longitud del vector con la información cifrada.

Por otro lado, en el ámbito de la criptografía, las claves de cifrado juegan un papel fundamental en la seguridad y confidencialidad de los datos. Estas claves son elementos secretos que sirven de guía a quien desencripta la imagen. En este caso, usaremos las llamadas claves simétricas, éstas utilizan las mismas clave tanto para cifrar como para descifrar los datos.
En este caso, nuestras claves de cifrado serán las siguientes:

- M= número de veces que aumentaremos la longitud del vector con la información original.
- a_1 y x_{01} = parámetro y condición inicial usados para generar la primera órbita.
- a_2 y x_{02} = parámetro y condición inicial usados para generar la segunda órbita.
- a_3 y x_{03} = parámetro y condición inicial usados para generar la segunda órbita.
- ubicacion inicial= posición inicial del vector información cifrada, se encuentra entre 0 y LM-1.

3.2.1. Algoritmo de cifrado.

Nuestro objetivo ahora es cifrar la imagen de la Figura 19.

Figura 19: Imagen original.

Esta imagen ha sido recogida de [6]. Posteriormente, la hemos modificado disminuyendo el número de píxeles de ésta con la finalidad de que el programa, que vamos a describir a continuación, pudiera encriptarla en un tiempo razonable.

- Paso 0: En primer lugar preparamos el programa añadiendo las librerías y las claves de cifrado correspondientes.

```
from PIL import Image
import numpy as np
from matplotlib import cm

#ClAVES DE CIFRADO:
M = 2
#Parametro y condicion inicial para la primera orbita
a_1=3.6
x_01=0.2
#Parametro y condicion inicial para la segunda orbita
a_2=3.7
x_02=0.5
#Parametro y condicion inicial para la tercera orbita
a_3=3.8
x_03=0.7

ubicacion_inicial = 0

def logistica(a,x): return a*x*(1-x) #Definimos la funcion logistica
```

- Paso 1: El primer paso para procesar una imagen es digitalizarla, lo cual implica almacenar cada uno de los subpíxeles en un vector que llamaremos Inf_original. Este vector tiene una longitud determinada L y su objetivo es conservar la información de la imagen en forma numérica para su posterior manipulación y análisis. Cada elemento del vector representa un subpíxel de la imagen, que corresponde a un componente específico de color rojo, verde o azul y están representados por un valor decimal entre 0 y 255.

```
#PASO 1:
Inf_original = Image.open('alhambra.jpg', 'r') #Cargamos la imagen
width, height = Inf_original.size #Obtenemos las medidas de la imagen
im = list(Inf_original.getdata()) #Sacamos los pixeles

#Lo ponemos todo en un solo vector
Inf_original = [item for sublist in im for item in sublist]

L = len(Inf_original) #Calculamos la longitud del vector Inf_original
```

- Paso 2: Como ya hemos dicho, el vector Inf_original está formado por L valores entre 0 y 255. En este paso, dividimos cada elemento del vector Inf_original entre 255 obteniendo así valores entre 0 y 1.

```
# Paso 2:
Inf_original = [x / 255 for x in Inf_original]
```

- Paso 3: A continuación, generamos el vector Inf_cifrada cuya longitud será LM. En este caso hemos decidido, por comodidad, que inicialmente todos los valores de este vector sean 2.

```
# Paso 3:
LM=L*M #Longitud de Inf_cifrada
Inf_cifrada = np.full(LM, 2) #Generamos el vector Inf_cifrada
Inf_cifrada= list(map(float, Inf_cifrada)) #Para que posteriormente pueda
    tomar valores decimales
```

- Paso 4: Mediante las claves de cifrado a_1 y x_{01} resolvemos la ecuación lógistica discreta L veces para dar lugar a la primera órbita. Los valores de dicha órbita los almacenaremos en el vector orbita_1.

```
# Paso 4:
orbita_1 = np.zeros((1,L))[0] #Generamos un vector de longitud L lleno de
    ceros
orbita_1[0] = x_01 #Inicializamos el primer valor del vector en x_01

for i in range(L-1):
    orbita_1[i+1] = logistica(a_1,orbita_1[i])
```

- Paso 5: Redondeamos cada valor del vector orbita_1 multiplicado por LM. Como resultado, obtendremos los valores que nos proporcionarán las posiciones que nos permitirán posteriormente aplicar la técnica de difusión.

```
# Paso 5:
posiciones = np.zeros((1,L))[0] #Generamos un vector de ceros de longitud L

for i in range(L):
    posiciones[i] = round(LM*orbita_1[i])
```

- Pasos 6-10 (Técnica de difusión): Esta técnica, como ya hemos explicado, consiste en recolocar los valores del vector Inf_original con la ayuda del vector posiciones generado en el paso anterior de la siguiente manera:

  - Paso 6: Asignamos a ubicacion el valor de ubicacion inicial que se será un entero entre 0 y $LM - 1$.

  - Paso 7: Calculamos la ubicación donde se colocarán los valores del vector Inf_original en el vector Inf_cifrada de la siguiente manera: ubicación= ubicación+ (posiciones[i] %L). La operación posiciones[i] %L se encarga de calcular las posiciones módulo L, por ello todos los valores del vector Inf_original estarán en un intervalor de longitud L en el vector Inf_cifrada.

  - Paso 8: Comprobamos si $ubicacion \leq LM$ y si es así se coloca en la ubicación de Inf_cifrada el valor de Inf_original. Si dicha ubicación ya está ocupada por un valor distinto de 2, entonces buscamos la siguiente posición vacía.

  - Paso 9: En el caso en el que $ubicacion > LM$, ubicacion=posiciones[i] %L y se coloca el valor de Inf_original en la nueva ubicación, en el caso de que esté ocupada buscamos nuevamente la siguiente posición vacía.

  - Paso 10: Realizamos los pasos 7 y 8 o 9 hasta colocar todos los elementos del vector Inf_original en el vector Inf_cifrada.

```
#TECNICA DE DIFUSION
# Paso 6: asignamos a ubicacion el valor de la clave de cifrado
    ubicacion_inicial
ubicacion = ubicacion_inicial

# Pasos 7-10: Metemos los valores del vector Inf_original en el vector
    Inf_cifrada desordenadamente en funcion del vector posiciones
#Estos valores estaran colocados en un intervalo de longitud L del vector
    Inf_cifrada ya que aplicamos el operador %

for i in range(0,L):
    #Paso 7
    ubicacion = int(ubicacion + (posiciones[i]%L))
    # Paso 8
    if(ubicacion <= LM):
        #Buscamos un hueco vacio
        while Inf_cifrada[ubicacion] != 2:
            ubicacion = ubicacion + 1
            if(ubicacion == LM): ubicacion = 0
        Inf_cifrada[ubicacion] = Inf_original[i]
    # Paso 9
    else:
        ubicacion = int(posiciones[i]%L)
        #Buscamos un hueco vacio
        while Inf_cifrada[ubicacion] != 2:
            ubicacion = ubicacion + 1
            if(ubicacion == LM): ubicacion = 0
        Inf_cifrada[ubicacion] = Inf_original[i]
```

Para entender mejor la técnica de difusión veamos la Figura 20 donde se aprecia visualmente como vamos colocando los valores del vector Inf_original en el vector Inf_cifrada mediante el vector posiciones. En la Figura 20, como en nuestro código, ubicación_inicial=0.

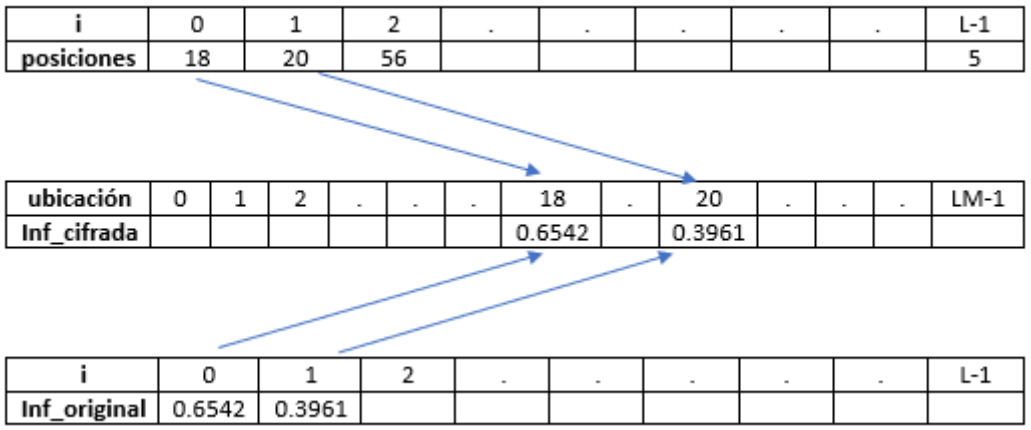

Figura 20: Técnica de difusión.

- Paso 11: En este paso, generamos la segunda órbita caótica. Para ello, resolvemos la ecuación logística $LM - L$ veces, obteniendo así el vector relleno.

```
# Paso 11:
relleno = np.zeros((1,LM-L))[0] #Generamos un vector de ceros de longitud LM-L
relleno[0] = x_02   #Inicializamos el vector relleno en x_02
for i in range(LM-L-1):
    relleno[i+1] = logistica(a_2,relleno[i])
```

- Paso 12: Recorremos el vector Inf_cifrada hasta encontrar una posición vacía, es decir, con el valor 2. Cuando se encuentra esta posición se introduce el primer valor del vector relleno. El proceso se repite hasta que todos los valores del vector relleno se hayan introducido en el vector Inf_cifrada, quedando este último relleno completamente.

```
# Paso 12:
ubicacion = 0
pos = 0
for number in Inf_cifrada:
    #Busco la posicion vacia
    if(Inf_cifrada[ubicacion] == 2):
         Inf_cifrada[ubicacion] = relleno[pos]
         pos = pos + 1
    ubicacion = ubicacion + 1
```

Como observamos en la Figura 21 terminamos de rellenar el vector Inf_cifrada con los valores del vector relleno.

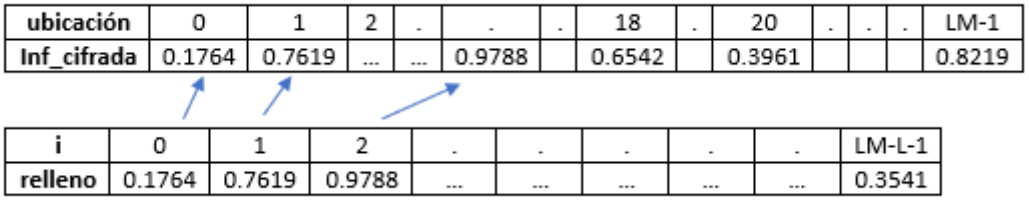

Figura 21: Rellenamos completamente el vector Inf_cifrada mediante el vector relleno.

- Paso 13-14 (Técnica de confusión): El fin de esta técnica es dispersar aún más los datos:
  - Paso 13: Resolvemos la ecuación logística LM veces generando así la tercera órbita cuyos datos los introduciremos en el vector confusión.
  - Paso 14: Sumamos, de forma ordenada, a cada valor del vector Inf_cifrada uno del vector confusión.

```
#TECNICA DE CONFUSION

# Paso 13:
confusion = np.zeros((1,LM))[0] #Generamos vector de ceros de longitud LM
confusion[0] = x_03 #Inicializamos el vector confusion en el valor x_03

for i in range(LM-1):
    confusion[i+1] = logistica(a_3,confusion[i])

# Paso 14:
#Mediante un bucle for sumamos, ordenadamente, a cada valor del vector
    Inf_cifrada un valor del vector confusion
for i in range(LM):
    Inf_cifrada[i] = Inf_cifrada[i] + confusion[i]
```

- Paso 15: Finalmente, solo faltaría convertir el vector Inf_cifrada en una imagen, la cual será nuestra imagen encriptada.

```
#Paso 15:Reconstruimos la imagen
factor2RGB = 255/max(inf_cifrada) #Le damos color
foto = []
fila = []
contador = 0
total_pixels = int(LM/3) #Como queremos un formato RGB el total de los pixeles
     es LM/3
ancho=320 #he cogido este ancho para que la imagen se quedara lo mas cuadrada
    posible 320.240=LM/3

for i in range(0,total_pixels):
    if (contador == ancho): #Verificamos si contador es igual a ancho. Si es
    verdadero, significa que se ha alcanzado el final de una fila de pixeles
    en la foto.
        print(fila)   #Mostramos los pixeles de la fila actual.
        foto.append(fila) #Agregamos la fila completa al vector foto
        fila = []  #Preparamos la lista para almacenar los pixeles de la
    siguiente fila.
        contador = 0 #Al completarse la fila se inicia el contador
    #Agregamos un nuevo pixel a la lista fila. Este pixel esta compuesto por
    tres valores obtenidos de la lista Inf_cifrada, multiplicados por
    factor2RGB.
    #Los valores se acceden en funcion del indice i y se utilizan los
    desplazamientos i*3, i*3+1 y i*3+2.
    fila.append((Inf_cifrada[i*3]*factor2RGB,Inf_cifrada[i*3+1]*factor2RGB,
    Inf_cifrada[i*3+2]*factor2RGB))
    contador = contador + 1

#Agregamos la ultima fila al vector foto
foto.append(fila)

#Convertimos el vector foto en una matriz
matriz_encriptada = np.array(foto, dtype=np.uint8)

#Creamos y guardamos nuestra imagen encriptada
imagen_encriptada = Image.fromarray(matriz_encriptada)

#Tomamos la matriz "matriz_encriptada" como argumento y crea un objeto de
    imagen a partir de ella.
imagen_encriptada.save('encriptada.jpg') #guardamos la imagen
```

Finalmente, nuestra imagen encriptada se muestra en la Figura 22.

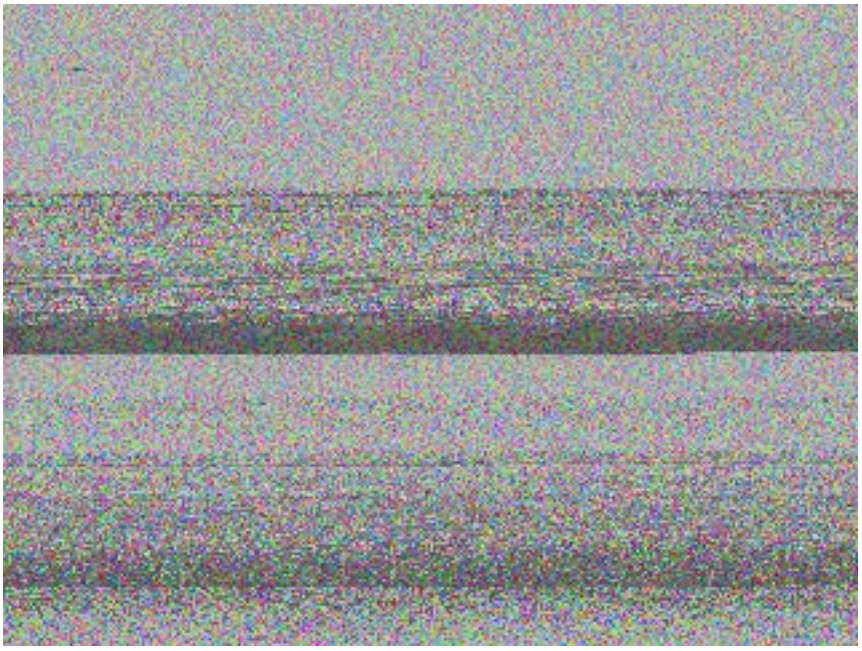

Figura 22: Imagen encriptada.

3.3. Descifrado.

El objetivo de la encriptación de una imagen es que en caso de atacante no se sepa de qué imagen se trata, pero quien la recibe debe ser capaz de desencriptarla para conseguir la imagen original y recibir así la información.

En este apartado desarrollaremos el código de desencriptación. Éste nos resultará muy sencillo una vez sabemos el código de cifrado ya que básicamente se trata de darle la vuelta a éste. Además, debemos partir de la siguiente información: el vector Inf_Cifrada, la longitud del vector Inf_cifrada y las claves de cifrado.

3.3.1. Algoritmo de descifrado.

- Paso 1: Generamos el vector confusión. Este paso coincide con el paso 13 del algoritmo de cifrado.

```
  # Paso 1:
confusion = np.zeros((1,LM))[0] #Generamos vector de ceros de longitud LM
confusion[0] = x_03 #Inicializamos el vector confusion en el valor x_03

for i in range(LM-1):
    confusion[i+1] = logistica(a_3,confusion[i])
```

- Paso 2: Eliminamos la técnica de confusión. Para ello restamos, de forma ordenada, a cada elemento del vector Inf_cifrada uno del vector confusion.

```
  # Paso 2:
for i in range(LM):
    Inf_cifrada[i] = Inf_cifrada[i] - confusion[i]
```

- Paso 3: Recreamos los vectores orbita_1 y posiciones. Es decir, los pasos 4 y 5 del algoritmo de cifrado.

```
  # Paso 3:
orbita_1 = np.zeros((1,L))[0] #Generamos un vector de longitud L lleno de
    ceros
orbita_1[0] = x_01 #Inicializamos el primer valor del vector en x_01
for i in range(L-1):
    orbita_1[i+1] = logistica(a_1,orbita_1[i])
posiciones = np.zeros((1,L))[0] #Generamos un vector de ceros de longitud L
for i in range(L):
    posiciones[i] = round(LM*orbita_1[i])
```

- Pasos 4-9: Eliminamos la técnica de difusión.

  - Paso 4: Reasignamos de nuevo a ubicación el valor de ubicacion_inicial.

  - Paso 5: Calculamos la ubicación de forma análoga al paso 7 del algoritmo de cifrado, es decir, ubicación= ubicación+ (posiciones[i] %L).

  - Paso 6: Comprobamos si $ubicacion \leq LM$ se toma el valor que hay en dicha ubicacion del vector Inf_cifrada. En el caso de que esa posición esté vacía se toma el valor de la siguiente posición no vacía.

  - Paso 7: Si $ubicacion > LM$ asignamos como en el paso 9 del algoritmo de cifrado a ubicacion un nuevo valor, ubicacion=posiciones[i] %L. Tras esto, se toma el valor de la nueva ubicación y, de nuevo, en caso de no encontrarse dicho valor se cogerá el de la siguiente posición no vacía.

  - Paso 8: Multiplicamos por 255 los valores obtenidos en los pasos anteriores y posteriormente los redondeamos. A continuación, almacenamos dichos valores ordenadamente en las posiciones vacías del vector Inf_original.

  - Paso 9: Repetimos los pasos anteriores hasta eliminar completamente la técnica de difusión. Finalmente, obtenemos el vector Inf_orignal relleno completamente con los datos originales.

```
  # Paso 4: Reasignamos a ubicacion el valor de ubicacion_inicial
ubicacion = ubicacion_inicial
# Paso 5: Creamos el vector Inf_original para posteriormente almacenar la
    informacion desencriptada
Inf_original = np.zeros((1,L))[0] #Vector de ceros de longitud L
#Pasos 6-9: Extraemos los valores de Inf_cifrada al vector Inf_original
for i in range(0,L):
    ubicacion = int(ubicacion + (posiciones[i]%L))
    # Paso 6
    if(ubicacion <= LM):
        # Buscamos un hueco vacio
        while Inf_cifrada[ubicacion] == 2:
            ubicacion = ubicacion + 1
            if(ubicacion == LM): ubicacion = 0
        Inf_original[i] = int(round(255*Inf_cifrada[ubicacion]))
        Inf_cifrada[ubicacion] = 2
    # Paso 7
    else:
        ubicacion = int(posiciones[i]%L)
        while Inf_cifrada[ubicacion] == 2:
            ubicacion = ubicacion + 1
            if(ubicacion == LM): ubicacion = 0
        Inf_original[i] = int(round(255*Inf_cifrada[ubicacion]))
        Inf_cifrada[ubicacion] = 2
Inf_original = Inf_original.astype(int) #Convertimos a valores enteros los
    datos del vector Inf_original
```

La Figura 23 nos muestra la realización de esta técnica.

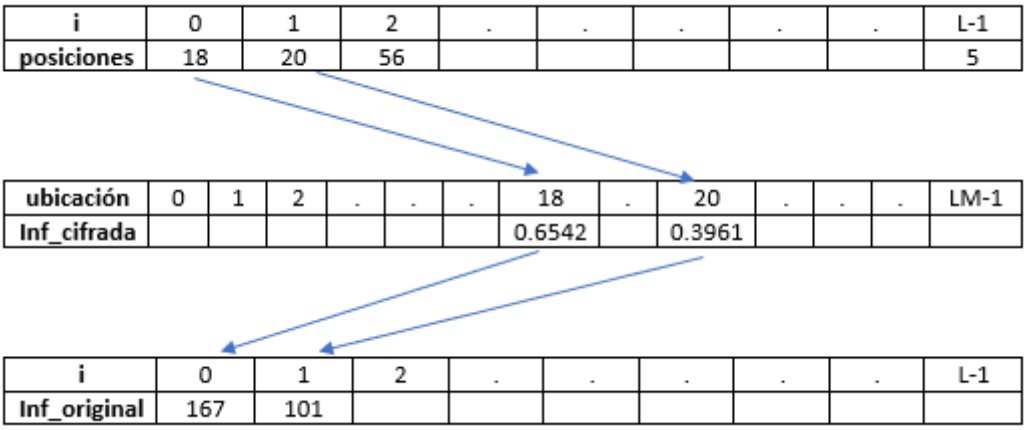

Figura 23: Eliminar difusión.

- Paso 10: Transformamos el vector Inf_original en una imagen.

  Los pasos a seguir en este proceso serán los siguientes: En primer lugar, creamos el vector foto, fila e inicializamos un contador a 0. Luego, mediante un bucle for vamos obteniendo filas y agregándolas al vector foto. Estas filas serán las filas de la matriz de píxeles que dará lugar a la imagen desencriptada. Finalmente, transformamos el vector foto en una matriz y a partir de ésta creamos la imagen desencriptada.

```
#Paso 10: Reconstruimos la imagen
foto = []
fila = []
contador = 0
total_pixels = int(L/3)

for i in range(0,total_pixels):
    if (contador == width): #Verificamos si contador es igual a ancho.
        foto.append(fila) #Agregamos la fila completa al vector foto
        fila = []  #Preparamos la lista para almacenar los pixeles de la siguiente fila.
        contador = 0 #Al completarse la fila se inicia el contador
    #Agregamos un nuevo pixel a la lista fila. Este pixel esta compuesto por tres valores obtenidos de la lista Inf_original.
    #Los valores se acceden en funcion del indice i y se utilizan los desplazamientos i*3, i*3+1 y i*3+2.
    fila.append((Inf_original[i*3],Inf_original[i*3+1],Inf_original[i*3+2]))
    contador = contador + 1

#Agregamos la ultima fila al vector foto
foto.append(fila)

#Convertimos el vector foto en una matriz
matriz_desencriptada = np.array(foto, dtype=np.uint8)

#Creamos y guardamos nuestra imagen encriptada
imagen_desencriptada = Image.fromarray(matriz_desencriptada)#Toma como argumento "matriz_desencriptada" y crea una imagen a partir de ella.

imagen_desencriptada.save('desencriptada.jpg') #Guarda la imagen
```

Tras la desencriptación, la imagen obtenida es la mostrada en la Figura 24.

Figura 24: Imagen desencriptada.

En efecto, podemos comprobar que la Figura 24 coincide con la Figura 19. Concluimos por tanto que la imagen se ha encriptado y desencriptado con éxito.

Pero, ¿es éste el método más rápido para la encriptación de una imagen? La respuesta es que no. Esto se debe a que el proceso de cifrado es muy lento debido a la técnica de difusión ya que estamos trabajando con un vector que tiene una longitud L muy grande. A la hora de realizar esta técnica, el programa tarda mucho en recorrer todo el vector hasta encontrar una posición vacía y además tiene que repetir el proceso L, lo que lo ralentiza mucho. Concretamente, esta imagen es una imagen de 240x160 píxeles, es decir, se trata de una imagen relativamente pequeña y, sin embargo, el programa tarda alrededor de unos 7 minutos en encriptarla y desencriptarla.

¿Habría entonces una forma más rápida para encriptar y desencriptar la imagen? El siguiente apartado nos da la respuesta a esta pregunta.

3.4. Aportación.

Como ya hemos comentado, aunque el algoritmo presentado anteriormente funcione a la perfección, lo que nosotras queremos es encontrar un algoritmo más eficiente. Esto es, que su desencriptación no sea fácil a los ataques, pero que a su vez su cifrado sea más rápido.

Para resolver este problema hemos pensado un nuevo algoritmo que actúa de la misma forma, pero escoge las posiciones de una manera diferente. Esta vez, la elección de las posiciones que ocuparán los elementos del vector Inf_original se hará de la siguiente forma:

Para obtener las posiciones, resolvemos la ecuación logística, LM veces, dando lugar a la primera órbita, la cual se almacenará en el vector orbita_1. A continuación, ordenamos los valores de este vector con sus respectivas posiciones originales, las cuales son diferentes entre sí y su valor es un entero entre 0 y $LM - 1$. Tras esto, asignamos al vector posiciones el valor de las posiciones originales así, éste estará relleno de LM valores enteros entre 0 y $LM - 1$. Como cada posición es diferente, añadimos los L valores del vector Inf_original en el vector Inf_cifrada de manera que las posiciones nos las darán los L primeros valores del vector posiciones. De esta forma, el nuevo algoritmo de cifrado será el siguiente:

3.4.1. Algoritmo de cifrado.

```
from PIL import Image
import numpy as np
from matplotlib import cm

#LLAVES DE CIFRADO:
M = 2
#Parametro y condicion inicial para la primera orbita
a_1=3.6
x_01=0.2
#Parametro y condicion inicial para la segunda orbita
a_2=3.7
x_02=0.5
#Parametro y condicion inicial para la tercera orbita
a_3=3.8
x_03=0.7 #este valor podra ser cualquiera del intervalo (0,1)

def logistica(a,x): return a*x*(1-x) #defino la funcion logistica

#NUEVO ALGORITMO DE ENCRIPTADO

#PASO 1: Digitalizar la imagen
Inf_original = Image.open('alhambra.jpg', 'r')   #Cargamos la imagen
width, height = Inf_original.size   #Obtenemos las medidas de la imagen
im = list(Inf_original.getdata())   #Sacamos los pixeles

# Lo ponemos todo en un solo vector
Inf_original = [item for sublist in im for item in sublist]

L = len(Inf_original)   #Calculamos la longitud del vector inf_original

# Paso 2: Dividimos entre 255 para que todos los valores se encuentre entre 0 y 1
Inf_original = [x / 255 for x in Inf_original]

#PASO 3: Generar el vector Inf_cifrada
LM=L*M  #Longitud de Inf_cifrada
Inf_cifrada = np.full(LM, 2)
Inf_cifrada= list(map(float, Inf_cifrada))  #Para que posteriormente pueda tomar
    valores decimales
```

A partir de aquí nuestro algoritmo cambia. La primera diferencia la observamos en el paso 4 ya que anteriormente resolvíamos la ecuación logística para obtener la primera órbita L veces mientras que ahora la resolvemos LM veces.

```
#PASO 4: Resolvemos la primera orbita mediante las llaves de cifrado a_1 y x_01.
orbita_1=np.zeros((1,LM))[0] #Generamos el vector de longitud LM lleno de ceros
orbita_1[0]=x_01    #Inicializamos el primer valor del vector en x_01

for i in range (LM-1):
    orbita_1[i+1]=logistica(a_1,orbita_1[i])
```

En el paso 5 es donde está la idea clave de la obtención de las posiciones. Si recordamos, en el anterior algoritmo obteniamos el vector posiciones multiplicando los valores del vector orbita_1 por LM y redondeando, posteriormente, dicho producto. El problema es que tras este redondeo el vector posiciones podía tener algunos valores repetidos. Sin embargo, en este caso lo que hacemos es ordenar en una tupla los LM valores del vector orbita_1, que obtuvimos en el paso anterior, junto con sus respectivas posiciones iniciales. Estas posiciones, evidentemente, son diferentes por lo que no se van a repetir. A continuación, separamos en dos listas independientes los valores ordenados del vector orbita_1 y los índices originales. Tras esto, generamos el vector posiciones de longitud LM que tomará los valores del vector indices_originales. Finalmente, rellenamos el vector Inf_cifrada de la siguiente manera: los L primeros elementos del vector posiciones, que son valores enteros entre 0 y $LM-1$, nos dará el lugar que ocuparán los elementos del vector Inf_original. Es decir, en la posición posiciones[0] de Inf_cifrada irá Inf_original[0], en la posiciones[1] de Inf_cifrada irá Inf_original[1] y así sucesivamente hasta introducir todos los valores de Inf_original en Inf_cifrada.

```python
#Paso 5: Realizamos el nuevo metodo para escoger las posiciones

#Sacamos la lista de tuplas (valor, indice)
valores_con_indices = [(valor, indice) for indice, valor in enumerate(orbita_1)]

#Ordenamos la lista de tuplas por los valores
valores_con_indices_ordenados = sorted(valores_con_indices)

#Obtenemos dos listas separadas con los valores ordenados y los indices originales
valores_ordenados = [tupla[0] for tupla in valores_con_indices_ordenados]
indices_originales = [tupla[1] for tupla in valores_con_indices_ordenados]

posiciones=np.zeros((1,LM))[0] #Generamos el vector de longitud LM lleno de ceros
posiciones=indices_originales  #El vector posiciones tomara los valores del vector
    indices_originales

for i in range (L):
    Inf_cifrada[posiciones[i]]=Inf_original[i]
```

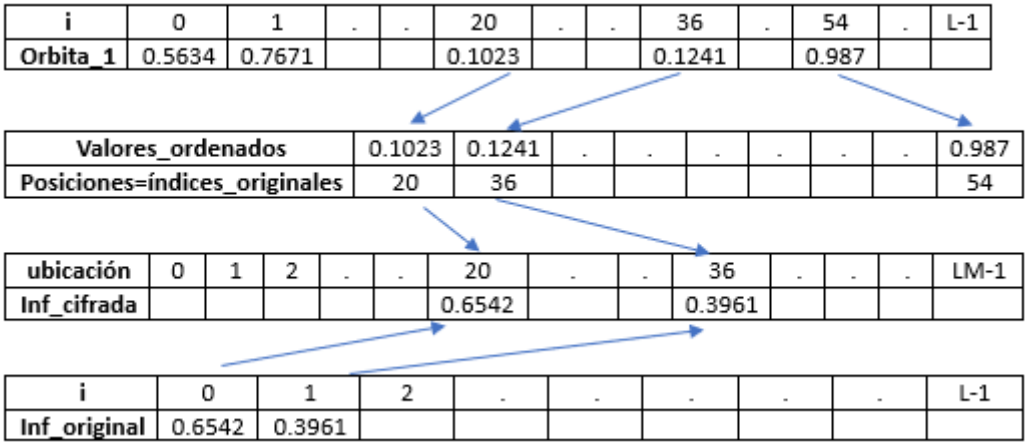

Figura 25: Obtención de las posiciones.

Los pasos que siguen hasta finalizar el algoritmo son análogos al algoritmo anterior.

```python
# Paso 6: Creacion del vector relleno
relleno = np.zeros((1,LM-L))[0] #vector de ceros de longitud LM-L
relleno[0] = x_02   #Inicializamos el vector relleno en x_02

for i in range(LM-L-1):
    relleno[i+1] = logistica(a_2,relleno[i])

# Paso 7: Terminamos de rellenar el vector Inf_cifrada con los datos del vector
    relleno
ubicacion = 0
pos = 0

for number in Inf_cifrada:
    if(Inf_cifrada[ubicacion] == 2):
         Inf_cifrada[ubicacion] = relleno[pos]
         pos = pos + 1
    ubicacion = ubicacion + 1

# Paso 8: Creamos el vector confusion
confusion = np.zeros((1,LM))[0] #Vector de ceros de longitud LM
confusion[0] = x_03 #Inicializamos el vector confusion en x_03

for i in range(LM-1):
    confusion[i+1] = logistica(a_3,confusion[i])

```

```
# Paso 9: Aplicamos la tecnica de confusion
for i in range(LM):
    Inf_cifrada[i] = Inf_cifrada[i] + confusion[i]
# Paso 10: # Reconstruimos la imagen
factor2RGB = 255/max(Inf_cifrada) # Para poder darle color
foto = []
fila = []
contador = 0
total_pixels = int(LM/3) #como queremos un formato RGB el total de los pixeles sera
     LM/3
ancho=320 #he cogido este ancho para que la imagen se quedara lo mas cuadrada
    posible 320.240=LM/3

for i in range(0,total_pixels):
    if (contador == ancho): #Verifica si contador es igual a ancho. Si es verdadero
    , significa que se ha alcanzado el final de una fila de pixeles en la foto.
        print(fila)   #Esto muestra los pixeles de la fila actual.
        foto.append(fila) # Esto agrega la fila completa al vector foto
        fila = []  # Esto prepara la lista para almacenar los pixeles de la
    siguiente fila.
        contador = 0 #al completarse la fila se inicia el contador
    #Agrega un nuevo pixel a la lista fila.Este pixel esta compuesto por tres
    valores obtenidos de la lista Inf_cifrada, multiplicados por factor2RGB.
    # Los valores se acceden en funcion del indice i y se utilizan los
    desplazamientos i*3, i*3+1 y i*3+2.
    fila.append((Inf_cifrada[i*3]*factor2RGB,Inf_cifrada[i*3+1]*factor2RGB,
    Inf_cifrada[i*3+2]*factor2RGB))
    contador = contador + 1

#Agrega la ultima fila al vector foto
foto.append(fila)
# Conviertimos el vector foto en una matriz
matriz_encriptada = np.array(foto, dtype=np.uint8)
# Creamos y guardamos nuestra imagen encriptada
imagen_encriptada = Image.fromarray(matriz_encriptada) #Toma la matriz "
    matriz_encriptada" como argumento y crea un objeto de imagen a partir de ella.
imagen_encriptada.save('encriptada.jpg')   #Guarda la imagen
```

Tras este nuevo algoritmo de cifrado la imagen que obtenemos ahora en la Figura 26 es diferente a la Figura 22, como cabía de esperar. Además se transformar en una imagen mucho más homogénea por lo que se dificulta el desencriptado ya que no da pistas sobre la imagen original.

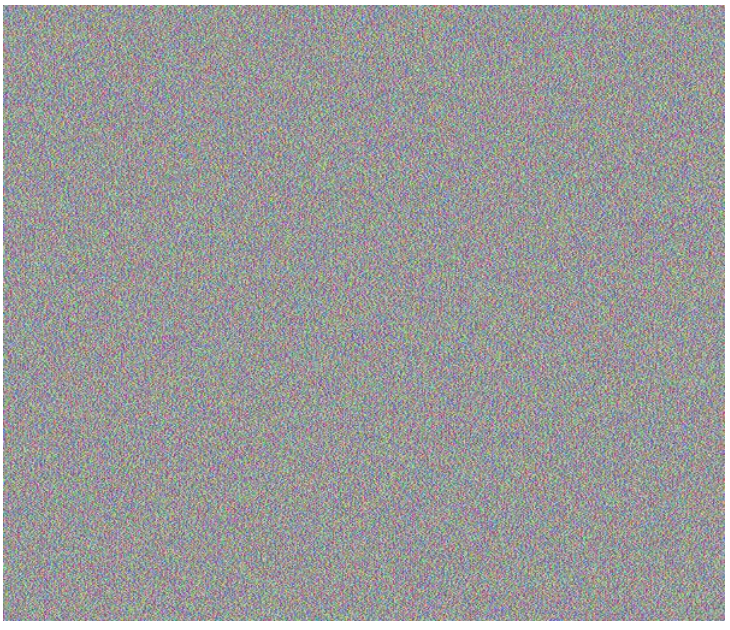

Figura 26: Imagen encriptada mediante otro algoritmo de cifrado.

Hemos comprobado tras la compilación de este nuevo algoritmo que, en efecto, se solucionan los problemas de lentitud que nos proporcionaba el anterior. Mientras que el primero tardaba alrededor de unos 7 minutos en encriptar y desencriptar la imagen éste no llega al minuto de compilación.

Si lo pensamos lo podríamos complicar todavía más. Es obvio que habrá infinitas formas de hacerlo, pero una idea es la siguiente: Si tomamos el valor $M = 8$ en vez de $M = 2$, al realizar el paso 5 se nos quedarán $7L$ posiciones vacías en el vector Inf_cifrada. Si en lugar de generar 3 órbitas en todo el algoritmo generamos 9 donde 7 de éstas tengan longitud L, podremos terminar de rellenar el vector Inf_cifrada al completo. Además, éste estará compuesto por valores muy aleatorios por lo que será un vector mucho más confuso y encontrar la relación de los valores será un trabajo muy tedioso para quien quiera atacar.

Tras esta breve aportación, a continuación pasamos a la desencriptación de la imagen. Como es evidente, al cambiar el algoritmo de cifrado también su algoritmo de descifrado será diferente.

3.4.2. Algoritmo de descifrado.

```
#ALGORITMO DE DESENCRIPTADO

# Paso 1: Recreamos confusion, es el paso 13 del algoritmo de cifrado
confusion = np.zeros((1,LM))[0] #Vector de ceros de longitud LM
confusion[0] = x_03 #Inicializamos el vector confusion en x_03

for i in range(LM-1):
    confusion[i+1] = logistica(a_3,confusion[i])

# Paso 2: Eliminamos la confusion, restando al vector Inf_cifrada el vector
    confusion
for i in range(LM):
    Inf_cifrada[i] = Inf_cifrada[i] - confusion[i]

#PASO 4: Coincide con el paso 4 del algoritmo de cifrado
orbita_1=np.zeros((1,LM))[0] #Generamos el vector de longitud LM lleno de ceros
orbita_1[0]=x_01    #Inicializamos el primer valor del vector en x_01

for i in range (LM-1):
    orbita_1[i+1]=logistica(a_1,orbita_1[i])
```

En el siguiente paso sacamos las posiciones donde se encuentran los valores que extraeremos al vector Inf_original.

```
#Paso 5: Sacamos las posiciones
#Sacamos la lista de tuplas (valor, indice)
valores_con_indices = [(valor, indice) for indice, valor in enumerate(orbita_1)]

#Ordenamos la lista de tuplas por los valores
valores_con_indices_ordenados = sorted(valores_con_indices)

#Obtenemos dos listas separadas con los valores ordenados y los indices originales
valores_ordenados = [tupla[0] for tupla in valores_con_indices_ordenados]
indices_originales = [tupla[1] for tupla in valores_con_indices_ordenados]

posiciones=np.zeros((1,L))[0] #Generamos el vector de longitud LM lleno de ceros
posiciones=indices_originales  #El vector posiciones tomara los valores del vector
    indices_originales
```

Tomamos, multiplicándolos por 255, los valores que ocupan las posiciones obtenidas en el paso anterior y los extraemos al vector Inf_original.

```
#PASO 6:
Inf_original=np.zeros((1,L))[0] #Generamos el vector de longitud L lleno de ceros

for i in range(L):
  Inf_original[i]=int(Inf_cifrada[posiciones[i]]*255)
```

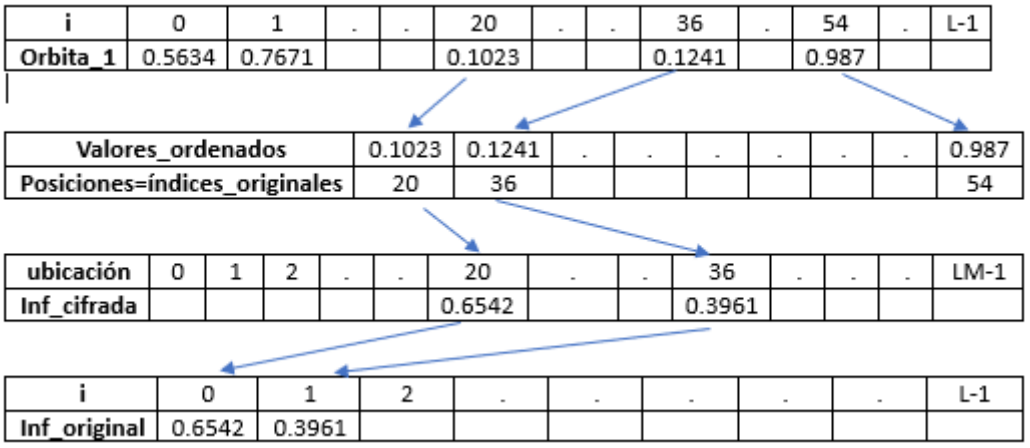

Figura 27: Pasos 5 y 6.

```python
#Paso 7: Reconstruimos la imagen
foto = []
fila = []
contador = 0
total_pixels = int(L/3)
for i in range(0,total_pixels):
    if (contador == width): #Verifica si contador es igual a ancho.
        foto.append(fila) # Esto agrega la fila completa al vector foto
        fila = []  # Esto prepara la lista para almacenar los pixeles de la siguiente fila.
        contador = 0 #Al completarse la fila se inicia el contador
    #Agrega un nuevo pixel a la lista fila.Este pixel esta compuesto por tres valores obtenidos de la lista Inf_original.
    # Los valores se acceden en funcion del indice i y se utilizan los desplazamientos i*3, i*3+1 y i*3+2.
    fila.append((Inf_original[i*3],Inf_original[i*3+1],Inf_original[i*3+2]))
    contador = contador + 1
#Agrega la ultima fila al vector foto
foto.append(fila)

# Convertimos el vector foto en una matriz
matriz_desencriptada = np.array(foto, dtype=np.uint8)

# Creamos y guardamos nuestra imagen encriptada
imagen_desencriptada = Image.fromarray(matriz_desencriptada)#Toma "como argumento " matriz_desencriptada" y crea una imagen a partir de ella.

imagen_desencriptada.save('desencriptada.jpg') #Guarda la imagen
```

Cuando desciframos mediante este algoritmo la Figura 26 obtenemos de nuevo la imagen de partida, Figura 19. Por tanto, podemos concluir que el proceso se ha realizado con éxito.

4. Capítulo 4: Propuesta de actividades de innovación para ESO y/o Bachillerato.

El caos es un fenómeno fascinante presente en algunos sistemas dinámicos, en particular, en la ecuación logística discreta. En esta sección, presentaremos algunas propuestas de actividades didácticas para enseñar y explorar el concepto de caos a través de la ecuación logística discreta. Estas actividades están diseñadas para ser implementadas en el ámbito educativo, específicamente en la asignatura de matemáticas a partir de 3^{o} ESO, con el objetivo de captar el interés del alumnado.

Como ya hemos visto en capítulos anteriores, la ecuación logística discreta es una ecuación en diferencias ampliamente estudiada en el campo de la dinámica de poblaciones y que se puede expresar como una sucesión en la cual cada término depende del término anterior. Lo que se pretende mostrar es que a pesar de su aparente sencillez, esta fórmula puede generar comportamientos caóticos, lo que implica una gran sensibilidad a las condiciones iniciales. Todo esto tiene relevancia en la vida cotidiana y en diversas áreas científicas aparte de las matemáticas, como son la biología, la física y la economía.

4.1. Desarrollo de actividades.

A continuación, propondremos algunas actividades, a modo divulgativo, con el fin de fomentar el interés del alumnado por las matemáticas y facilitar la compresión del comportamiento de la ecuación logística discreta.

En primer lugar, para captar la atención del estudiantado, se mostrará una imagen cualquiera como la que vemos en la Figura 28.

Figura 28: Imagen inicial.

Si dicha actividad se está desarrollando en un curso de ESO, se les contará a los alumnos que una imagen se trata de una colección de números, los cuales indican el color de cada píxel que forma la imagen.

En el caso de que esta actividad se desarrolle en un curso de bachillerato, se les explicará brevemente que una imagen no es más que una matriz de números, donde el número de columnas es 3 y cada columna se corresponde con los colores rojo, verde y azul, respectivamente.

Tras esto, se mostrará al alumnado la Figura 29 y se les preguntará si creen que esta nueva imagen tiene alguna relación con la anterior. Tras esto, se les hará entender que debajo de este proceso, llamado *proceso de encriptación*, hay matemáticas. Para ello, explicaremos brevemente y de forma general que hemos partido de una imagen inicial y para llegar a la imagen encriptada hemos decolocado los píxeles de la original y añadido algunos nuevos.

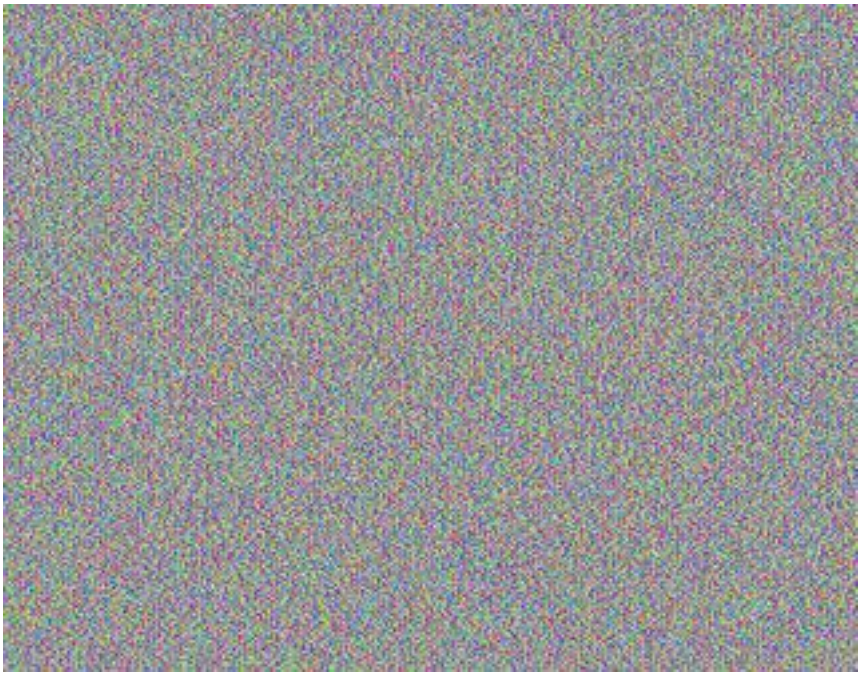

Figura 29: Imagen encriptada.

A continuación, para que se pueda comprender un poco más de que forma actúan las matématicas en la encriptación de la imagen, hablaremos de la ecuación logística discreta asociándola con una sucesión.

Mostraremos la forma de la ecuación logística discreta:

$$x_{k+1} = ax_k(1 - x_k), \qquad 0 \leq a \leq 4, \tag{4.1}$$

y haremos entender cómo a partir de un término inicial x_0, que toma valores entre 0 y 1, se pueden obtener los siguientes .

Por último, finalizaremos la actividad presentando el efecto mariposa. Explicaremos cómo para valores de a entre 3.57 y 4, aparece un fenómeno denominado *caos*. Para que se entienda mejor este concepto, nos ayudaremos de algunas simulaciones y visualizaciones iteractivas que muestren el caos, como por ejemplo:

https://www.geogebra.org/classic/hyFXGQWk
https://www.geogebra.org/m/ktkzunk7#material/bqtze5aj

También podemos mostrar visualmente, mediante la representación gráfica de las soluciones, cómo para un valor de a entre 3.57 y 4, pequeños cambios en las condiciones iniciales cambian completamente el comportamiento de las soluciones.

Por ejemplo, para $a = 4$, una condición inicial $x_0 = 0{,}3$ y otra $x_0 = 0{,}30000001$ la Figura 30 nos muestra cómo ese insignificante cambio en la condición inicial produce un gran cambio en la solución de la ecuación.

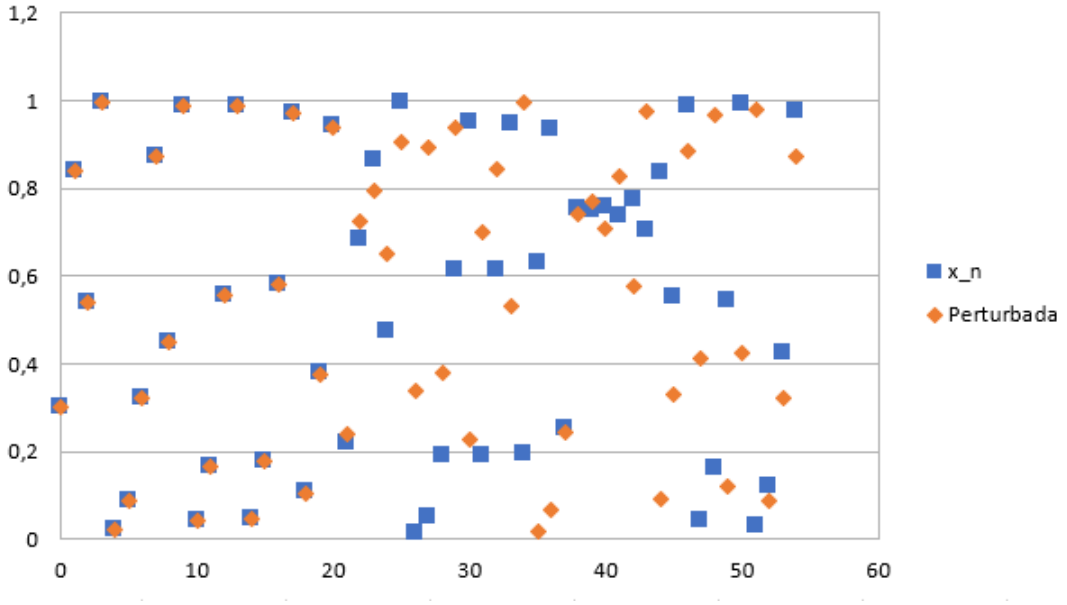

Figura 30: Representación gráfica del efecto mariposa.

Sin embargo, si para las mismas condiciones iniciales tomamos $a = 2$, podemos observar en la Figura 31 que no se produce ese radical cambio en la solución.

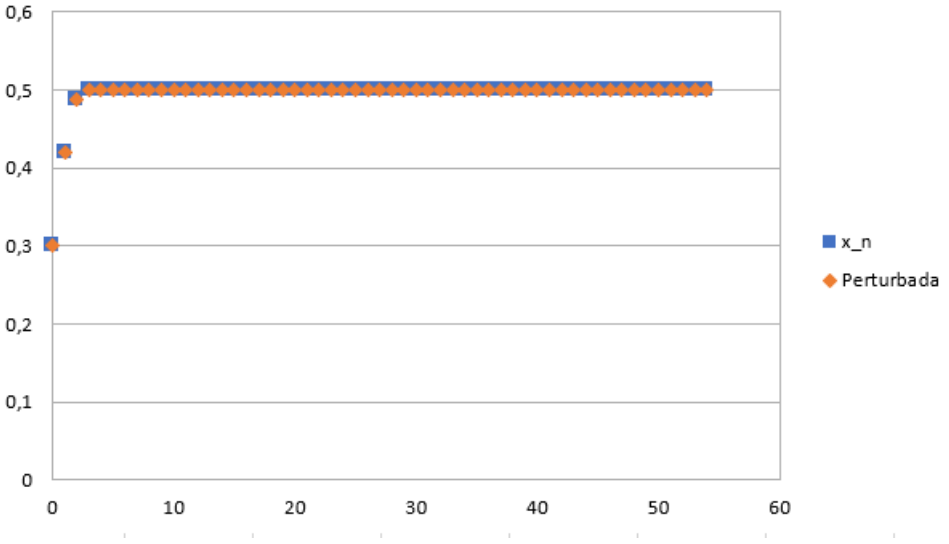

Figura 31: Representación gráfica de las soluciones de la ecuación logística discreta para pequeños cambios en sus condiciones inciales, con $a = 2$.

La propuesta de actividades presentada en este trabajo pretende ofrecer a quien enseña, una idea práctica para abordar el tema del caos a través de la ecuación logística discreta. Esperamos que estas actividades motiven al alumnado a explorar y comprender la complejidad del caos y a entender cómo las matemáticas se encuentran donde menos lo esperan.

Conclusiones.

En este Trabajo de Fin de Grado, hemos analizado la relación entre la ecuación logística discreta y su aplicación en el campo de la criptografía.

La principal conclusión obtenida y a destacar es que la ecuación logística discreta ha demostrado ser una herramienta matemática potente para generar secuencias pseudoaleatorias, ya que ofrece una gran sensibilidad a las condiciones iniciales que hace que las secuencias generadas sean impredecibles, lo que es esencial para la seguridad criptográfica.

Por otro lado, durante la realización de mi Trabajo de Fin de Grado, he tenido la oportunidad de poder combinar dos ramas que me encantan: las matemáticas y la informática.

Desde el principio, me he encontrado con algunas dificultades. En primer lugar, he tenido que estudiar a fondo las ecuaciones en diferencias ya que durante la asignatura impartida en el grado, Modelos Matemáticos I, esta parte no pudo ser vista detalladamente.

El siguiente inconveniente apareció al realizar el capítulo 3, pues mi idea desde el primer momento era programar en *C++*, que es el lenguaje de programación que se nos ha enseñado durante el grado. Pero, tras unas semanas sin éxito y frustación, por problemas relacionados con la instalación de librerías, decidí dar el paso de implementar dicho algoritmo en *Python*, lenguaje de programación con el que yo nunca había trabajado. Otro obstáculo fue la lentitud del primer algoritmo, ya que las imágenes a encriptar tenían que ser de muy pocos píxeles en comparación con los que normalmente tiene una imagen, pues sino podía tardar horas y horas en compilar. Es por ello, que a pesar de no formar parte de los objetivos del trabajo, decimos pensar un nuevo algoritmo que nos proporcionara una mayor velocidad. Tras varías reuniones con mi tutora y días dándole vueltas se nos ocurrió una idea que parecía mucho más eficiente. Para nuestra sorpresa, después de su implementación vimos que además de ser mucho más rápido, el programa nos devolvía una imagen encriptada mucho más homogénea que la que nos proporcionaba el primero, lo cual nos indicó que se trataba de una buena propuesta.

Otro punto a destacar es que se podrían incluir algunas mejoras, por ejemplo en el capítulo 3 cuando hablamos de claves de cifrado, estas claves no se dan tal cual pues podría dar pistas a quien quiera atacar. Una forma que se me ocurre de dar las claves es, por ejemplo, asociándolas a las letras del abecedario, es decir, asociar los números del 0 al 9 y el punto que separa la parte decimal de la entera con letras. Además, respecto a nuestra aportación, ya dijimos una manera para proporcionar aún más seguridad al programa, que consistía en lo siguiente: Podíamos tomar, por ejemplo, un valor de M=8 y así, al realizar el paso 5, se nos quedarían 7L posiciones vacías que podrían ser rellenadas por 7 órbitas de longitud L.

Para finalizar con las conclusiones mencionar que, en general, mi experiencia al realizar el TFG a pesar de haberme traído algún que otro dolor de cabeza, también me he aportado muchas cosas buenas. Por ejemplo, he podido trabajar con un lenguaje de programación que nunca había manejado, he aprendido a aplicar mis conocimientos teóricos adquiridos a lo largo de estos cuatro años de una forma distinta a la que estaba acostumbra, que era mediante la realización de exámenes. Además, mi interés por el tema de la criptografía y la programación se ha profundizado aún más.

Por último, quiero expresar mi profundo agradecimiento a mi tutora del proyecto, María José, sin su orientación este trabajo no hubiera sido posible. No puedo dejar tampoco de mencionar a mi familia y amigos, por su paciencia y apoyo a lo largo de toda mi carrera universitaria y por haber confiado en mí, incluso cuando ni yo misma lo hacía.

Bibliografía

[1] URL: https://personal.us.es/jmiguel/MATECO/Web-Matheco.pdf.

[2] URL: https://www.eco.uc3m.es/~rimartin/Teaching/AMATH/NOTES2SP.pdf.

[3] URL: https://www.geogebra.org/m/uvsfvNDt.

[4] URL: https://www.programafacil.org/2021/01/la-ecuacion-del-caos.html.

[5] URL: https://github.com/pedrocadiz/EncriptacionOrbitasCaoticas.

[6] URL: https://www.spain.info/export/sites/segtur/.content/imagenes/cabeceras-grandes/andalucia/alhambra-granada-20044065-istock.jpg_1014274486.jpg.

[7] URL: https://www.geogebra.org/classic/hyFXGQWk.

[8] URL: https://www.geogebra.org/m/ktkzunk7#material/bqtze5aj.

[9] S Elaydi. *Discrete Chaos, Chapman & Hall/CRC*. 2000.

[10] Carlos Fernández Pérez, Francisco José Vázquez Hernandez y José Vegas Montaner. *Ecuaciones diferenciales y en diferencias: sistemas dinámicos*. Ediciones Paraninfo, SA, 2003.

[11] María J. Cáceres Granados. *Algunos apuntes sobre el Tema 3 de la asignatura Modelos Matemáticos y Algoritmos*.

[12] Tien-Yien Li y James A. Yorke. *Period Three Implies Chaos*. 1975.

[13] Jiménez-Rodrıguez Maricela, Flores-Siordia Octavio y González-Novoa Marıa Guadalupe. "Sistema para codificar información implementando varias órbitas caóticas". En: *Ingenierıa, investigación y tecnologıa* 16.3 (2015), págs. 335-343.

[14] James D Murray. *Mathematical biology II: Spatial models and biomedical applications*. Vol. 3. Springer New York, 2001.

[15] Teresa E. Pérez y Miguel A. Piñar. *La ecuación lineal en diferencias*.

[16] Rafael Ortega Rıos. *Modelos matemáticos*. Editorial Universidad de Granada, 2013. Cap. 1.

[17] Ernesto Salinelli y Franco Tomarelli. *Discrete dynamical models*. Vol. 76. Springer, 2014.

[18] Juan Navas Ureña. *Sistemas Dinámicos Discretos*. Cap. 10.