UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO E IMPLEMENTACIÓN DE UNA APLICACIÓN INFORMÁTICA CON MATLAB PARA LA ENCRIPTACIÓN FRACTAL DE IMÁGENES BASADA EN EL CIFRADO DE HILL

AUTOR: ALBERT VIGUER GÜÉMEZ

TUTOR: FERNANDO GIMÉNEZ PALOMARES

Curso Académico: 2019-20

Agradecimientos

Quiero agradecer, en primer lugar, el apoyo de mis padres, José Vicente y Rosa, sin los cuales la realización de este proyecto no habría sido posible, así como al resto de mi familia por su interés y apoyo.

A mi tutor, Fernando Giménez, que fue quien me propuso el proyecto y su desarrollo en primer lugar, y sin su inestimable ayuda, aporte académico al trabajo, así como su gran conocimiento de la materia tratada, este proyecto hubiese permanecido en el abstracto. Ha sido un placer trabajar con él de principio a fin.

A mis compañeros de estudios, que han sido de gran apoyo durante el trascurso del proyecto, en especial a Ignacio Desco, por la ayuda moral y consejo aportados durante el desarrollo de este proyecto.

Por último y en general, a todos mis amigos y conocidos que han mostrado interés, apoyo y admiración hacia el trabajo y su desarrollo.

Resumen

DESARROLLO E IMPLEMENTACIÓN DE UNA APLICACIÓN INFORMÁTICA CON MATLAB PARA LA ENCRIPTACIÓN FRACTAL DE IMÁGENES BASADA EN EL CIFRADO DE HILL

En este trabajo final de grado se presenta una novedosa aplicación del álgebra matricial y la aritmética modular al proceso de encriptación de imágenes, a partir de los procesos iterativos en el plano complejo que dan lugar a la generación de fractales, aplicados al algoritmo de cifrado de Hill. Complementariamente, se pretende usar el protocolo de criptografía asimétrica Diffie-Hellman, basado en el problema del logaritmo discreto, para el establecimiento de claves entre las partes que van a compartir la información. Se pretende crear una herramienta informática basada en métodos numéricos, usando el entorno de desarrollo integrado MATLAB para implementar la aplicación.

En primer lugar, hay que describir con detalle y de manera rigurosa los métodos de encriptación que se utilizarán. En segundo lugar, hay que desarrollar el software para cada uno de los métodos descritos y, finalmente, presentar y estudiar los resultados obtenidos por ensayo empírico y utilizando las técnicas estadísticas más usuales para la comprobación de uniformidad y aleatoriedad, lo que permite determinar la fiabilidad de la metodología utilizada. Entre dichas técnicas se encuentran tablas de frecuencias e histogramas, las pruebas de Chi-cuadrado, de Kolmogorov-Smirnov, la prueba de correlación serial, pruebas de cantidad de rachas crecientes y decrecientes y bajo y sobre la media, y pruebas de longitud de rachas.

El objetivo final del proyecto es desarrollar la aplicación, exponer los algoritmos creados, ensayarlos y analizar y valorar su efectividad.

Palabras clave: Métodos numéricos, algoritmo, fractales, encriptación de imágenes, criptografía asimétrica, álgebra matricial, MATLAB, aplicación

Abstract

DEVELOPMENT AND IMPLEMENTATION OF A SOFTWARE APPLICATION WITH MATLAB FOR IMAGE ENCRYPTION WITH FRACTAL METHODS BASED IN HILL CIPHER

In this Project we are presenting an innovative application of matrix algebra and modular arithmetic to the process of image encryption, stemming from iterative processes in the complex plane that lead to the generation of fractals applied to the Hill cipher algorithm. Additionally, we are employing the Diffie-Hellman protocol of asymmetrical cryptography, based in the discrete logarithm problem, to establish the encryption key between the parts that are sharing the information. The intention is to create a computational tool based in numerical methods, using the integrated development environment MATLAB to implement the app.

In the first place, we are describing rigorously and in detail the encryption methods utilised. Secondly we shall develop applied software for each described method and finally we will present and study the obtained results by empirical assay and using common statistical techniques, to check for uniformity and randomness in results, which allows to determine the reliability of the employed methodology. Among these techniques we have frequency tables and histograms, Chi squared tests, Kolmogorov-Smirnov tests, serial correlation tests, increasing and decreasing streak tests, streaks above and below mean tests and streak length tests.

The ultimate aim of the Project is to develop the app, present the algorithms created and evaluate, analyse and assess their effectiveness.

Keywords: Numerical methods, algorithm, fractals, image encryption, asymmetrical cryptography, matrix algebra, MATLAB, application

Resum

DESENVOLUPAMENT I IMPLEMETACIÓ D'UNA APLICACIÓ INFORMÀTICA AMB MATLAB PER A LA ENCRIPTACIÓ FRACTAL D'IMATGES BASADA EN EL XIFRAT DE HILL

En aquest treball de fi de grau es presenta una nova aplicació de l'àlgebra matricial i l'aritmètica modular al procés d'encriptació d'imatges, a partir dels processos iteratius en el pla complex que donen lloc a la generació de fractals aplicats a l'algorisme de xifrat de Hill. Complementàriament, es pretén usar el protocol Diffie-Hellman de criptografía asimètrica, basat en el problema del logaritme discret, per a l'establiment de claus entre les parts que compartiran la informació. Es pretén crear una eina informática basada en métodoes numérics, usant l'entorn de desenvolupament integrat Matlab per a implementar l'aplicació.

En primer lloc, cal descriure amb detall i de manera rigorosa els mètodes d'encriptació que s'utilitzaran. En segon lloc cal desenvolupar el programari per a cadascun dels mètodes descrits i finalment presentar i estudiar els resultats obtinguts per assaig empíric i utilitzant les tècniques estadístiques més usuals per a la comprovació d'uniformitat i aleatorietat, la qual cosa permet determinar la fiabilitat de la metodologia utilitzada. Entre aquestes tècniques es troben taules de freqüències i histogrames, els test de la Chi quadrat, de Kolmogorov-Smirnov, la prova de correlació serial, proves de ratxes creixents i decreixents, proves de ratxes baix i sobre la mitjana i proves de longitud de ratxes.

L'objectiu final del projecte és desenvolupar l'aplicació, exposar els algorismes creats, testar-los i analitzar i valorar la seua efectivitat.

Paraules clau: Métodes numérics, algorisme, fractals, encriptació d'imatges, criptografia asimètrica, àlgebra matricial, Matlab, aplicació

Índice general

MEMORIA

PRESUPUESTO

Listado de figuras

Listado de tablas

Parte 1

MEMORIA

1. Introducción

1.1 Motivación y objetivos

El tratamiento seguro de información es un campo de gran importancia en la era digital en la que vivimos, y en especial en ambientes industriales, ya que, por ejemplo, las empresas de industria de alta gama buscan protección contra espionaje industrial, filtraciones y demás tipos de fraude relacionados con el intercambio de información. Es indudable que los sistemas de seguridad que existen en Internet son ciertamente fiables, los mecanismos de infraestructura de clave pública implementados en los sistemas informáticos permiten la ejecución de operaciones de cifrado y firma digital que aseguran a los receptores de información la autenticidad del mensaje enviado y la verificación del emisor. El sistema criptográfico en clave pública más extendido es el RSA desarrollado en 1977, y, más recientemente, el algoritmo Rijndael fue escogido por concurso como el *Advanced Encription Standard* y está empezando a ser implementado masivamente. [1, 2]

Aunque el envío de información por Internet es seguro, a priori, siempre se pueden añadir capas de seguridad agregadas que dificulten exponencialmente la posibilidad de obtención de esta información a un interceptor, específicamente cuando los datos enviados son imágenes, como puede ser el caso en la industria con planos, imágenes de prototipos o maquetas, escaneos de presupuestos o listas, etc. El objetivo de este proyecto es fundamentalmente este, desarrollar un sistema informático de codificación y decodificación de imágenes, que puedan ser enviadas por medios típicos junto con la clave de cifrado. El proyecto abordará el desarrollo de un programa que lea imágenes y las trasponga en matrices de rojos azules y verdes y las codifique a tramos por cifrado de Hill con matrices pseudoaleatorias generadas con mecanismos recurrentes de generación de fractales, obtendremos más programas de cifrado variando la frecuencia de generación de matrices y agregando un paso de permutación de valores. Las operaciones de generación de matrices pseudoaleatorias dependen de una serie variables definidas por el usuario cuyo conocimiento es requerido para realizar la operación inversa, de decodificación, y por tanto conforman la clave de cifrado, y, además, agregaremos una etapa previa complementaria de cifrado en clave pública con el algoritmo de Diffie-Hellman que hace al sistema entero muy robusto ante ataques externos por fuerza bruta.

Con todo, los objetivos principales del proyecto serán:

- El desarrollo de un sistema de cifrado de imágenes fiable y versátil
- La exploración de la herramienta MATLAB y las posibles aplicaciones que ofrece.
- El estudio del campo matemático de la criptografía.
- La aplicación de algoritmos matemáticos de cifrado a ambientes industriales.

1.2 Antecedentes

La criptografía es un campo con mucha más historia de la que se le supone comúnmente, ya que la necesidad de codificar y compactar la información ha sido una inquietud humana desde que se desarrollaron los propios medios de intercambio de información. Ya en el siglo V a.C. los éforos de Esparta utilizaban un método de cifrado por trasposición conocido como 'Escítala' para encriptar órdenes militares en campañas, basado en escritura sobre una vara de un cierto grosor, siendo necesaria otra vara igual para leer el mensaje claro. En la antigua Roma se utilizaba el cifrado César, un cifrado básico por sustitución nombrado por el propio Julio César, que lo utilizó frecuentemente. La encriptación de mensajes se siguió utilizando a través de la historia tanto en ambientes militares como para cifrar mensajes personales o órdenes políticas en las sociedades más desarrolladas de Europa, Oriente Medio y el Lejano Oriente, hasta que nos encontramos con la máquina Enigma, uno de los sistemas de cifrado más célebres de la historia. Enigma era el nombre que recibía la máquina de cifrado que utilizaron los alemanes en la Segunda Guerra Mundial para cifrar órdenes enviadas a largas distancias cuando existía riesgo de intercepción por parte receptores de radio aliados. Se basaba en un complejo sistema de rotores que relacionaban el carácter introducido en la máquina con uno aparentemente aleatorio a la salida. [1]

Una clase de métodos de cifrado son los métodos por sustitución, en los que las unidades del texto claro ya sean caracteres o valores, son sustituidas por nuevas unidades, de acuerdo con un sistema regular (a diferencia por ejemplo del cifrado por trasposición donde las unidades son permutadas entre ellas). Una distinción entre métodos de cifrado por sustitución es entre monoalfábéticos y polialfabéticos. En los primeros, una unidad es sustituida por otra correspondiente, que es siempre la misma, mientras que en los segundos una cierta unidad no se sustituye siempre por la misma. Ejemplos de métodos polialfabéticos son el cifrado de Alberti, el cifrado de Vignère y el cifrado de Hill, este último será enfocado en este proyecto.[5] Volviendo a la historia, la criptografía dio un salto con el desarrollo del cifrado asimétrico o cifrado en clave pública, el cifrado simétrico o en clave privada es el más simple, con una misma clave para cifrar y para descifrar, mientras que el cifrado asimétrico hace uso de dos claves, una pública, conocida por todos los participantes, y una privada, única para cada participante que sirve para descifrar lo que cifra la pública, ofreciendo una capa extra de seguridad que lo hace mucho más seguro. Esto permitió desarrollar sistemas esenciales en la codificación digital actual, como, por ejemplo, la firma digital. [1, 3]

Figura 1. Escítala del siglo V a.C.

Figura 2. Máquina Enigma

1.3 Estructura

La estructura de trabajo que seguiremos es la siguiente: en primer lugar, desarrollaremos el programa principal *codefoto1* toma una imagen en claro, la separa en las matrices de rojos, azules y verdes, y la codifica por tramos por medio del cifrado de Hill, utilizando matrices codificadoras pseudoaleatorias generadas por métodos fractales, con otro programa llamado *genemat*, generando matrices nuevas en cada iteración para el cálculo de cada tramo, así como el correspondiente programa de descifrado, que hemos llamado *decodefoto1* que realiza las mismas etapas que *codefoto1* en sentido inverso. Desarrollaremos también programas semejantes al primero como *codefoto2* que calcula únicamente una matriz y un vector pseudoaleatorios y realiza todo el cifrado con estos, con su respectivo *decodefoto2*, así como *codefoto3* que al procedimiento de *codefoto2* añade una permutación de píxeles previa en las matrices, intercambiando sus posiciones de acuerdo con vectores de redireccionamiento, junto con *decodefoto3*. Posteriormente introduciremos y programaremos el algoritmo de Diffie-Hellman que se implementa con cuatro programas que se ejecutan en el orden de intercambio de claves compartidas, previamente al cifrado y envío de la foto, de forma que tras ejecutar cada uno por parte del emisor o receptor se realiza un envío de datos al otro usuario, que el otro usuario utiliza para ejecutar el siguiente programa en la secuencia. Al final ambos usuarios comparten sus claves públicas con el otro y pueden calcular la clave privada compartida que es la que utilizarán. [4, 5]

Por tanto, el algoritmo de Diffie-Hellman se puede utilizar para encriptar en clave pública uno de los parámetros de la clave principal de los programas de codificación principales, aunque se puede utilizar para codificarlo más de uno e incluso todos para más seguridad, o ninguno, ofreciendo la posibilidad de cifrar en clave pública o privada. A continuación, realizaremos una serie de ensayos de los programas y unas pruebas estadísticas para demostrar la validez del algoritmo principal, el de *codefoto1*, y evaluar su uniformidad e independencia. Finalmente embeberemos los programas a un formato de aplicación de Matlab, así como a un ejecutable y una aplicación web, con una interfaz muy intuitiva e interactiva para el usuario de forma que sea muy atractiva.

Todo este trabajo se desarrollará en Matlab, una herramienta de programación matemática puntera en ambientes académicos y de investigación [6]. El entorno de Matlab, que es un acrónimo para *Matrix Laboratory*, "Laboratorio de matrices" en castellano, permite programar algoritmos y operaciones matemáticas, vectoriales y matriciales, de gran complejidad con un lenguaje de programación propio. Más allá del sistema base de cálculo y programación, Matlab cuenta con múltiples aplicaciones y *add-ons*, uno de ellos es el *app designer* [7] con el que hemos diseñado una interfaz de usuario para los programas de codificación y generado las aplicaciones de Matlab interactivas con las que el usuario puede realizar la codificación con sencillez.

2. Trabajo teórico

2.1 Aritmética modular

Las operaciones matriciales que desarrollaremos en los algoritmos de encriptación se llevarán a cabo en módulo m, en el espacio $\mathbb{Z}/m\mathbb{Z} = \{0,1,2,\dots,m-1\}$, también denominado $\mathbb{Z}_m$, que es el término que utilizaremos de aquí en adelante, que designa al conjunto de posibles restos del cociente de un entero entre el módulo m. Trabajaremos por tanto en un sistema de aritmética modular, que es aquel que se construye sobre la relación de congruencia entre enteros, es decir, si trabajamos en módulo m, cualquier número a módulo m es igual a el resto del cociente de a entre m, por ejemplo: $81 \bmod 7 = 4$. Sobre aritmética modular puede consultarse la referencia: [8]

La función `mod` nos permite implementar este cálculo en Matlab, y su notación es la siguiente: `mod(81,7)=4`. De forma ilustrativa, se puede escribir un programa en Matlab que nos proporcione la tabla de multiplicar de un conjunto en $\mathbb{Z}_m$, con mod m dado:

```
function A = tablam(m)
A =zeros(m+1);
A(1,2:m+1) = 0:m-1;
A(2:m+1,1) = 0:m-1;
for i=2:m+1
    for j=2:m+1
        A(i,j) = mod((i-2)*(j-2),m);
    end
end
A(1,1) = m;
```

Así, si ejecutamos para m=7:

```
tablam(7)
ans =
   7   0   1   2   3   4   5   6
   0   0   0   0   0   0   0   0
   1   0   1   2   3   4   5   6
   2   0   2   4   6   1   3   5
   3   0   3   6   2   5   1   4
   4   0   4   1   5   2   6   3
   5   0   5   3   1   6   4   2
   6   0   6   5   4   3   2   1
```

7	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Figura 3. Tabla de multiplicar de $\mathbb{Z}_7$

Se obtiene una matriz A donde el término a_{11} es el valor de m, el resto de la primera fila y la primera columna son los valores del espacio $\mathbb{Z}_m$, es decir, todos los números naturales de 0 a m-1: $\mathbb{Z}_m = \{0,1,2,\dots,m-1\}$, y el resto de las entradas son el producto del natural de cada fila y columna mod m.

Además, en $\mathbb{Z}_m$ se cumplen las propiedades:

$$a\ (mod\ m) + b(mod\ m) = (a+b)(mod\ m) \quad [2.1]$$

$$a\ (mod\ m) \cdot b(mod\ m) = (a \cdot b)(mod\ m) \quad [2.2]$$

De esta forma, las operaciones matriciales que llevaremos a cabo en la generación de matrices pseudoaleatorias y el cifrado de Hill en $\mathbb{Z}_m$ obedecen estas propiedades, lo cual agiliza enormemente el cálculo que es en muchos casos repetición sistemática.

El inverso de cualquier valor en $\mathbb{Z}_m$ es el número natural que multiplicado por este de comom resultado 1 en $\mathbb{Z}_m$, por ejemplo, el inverso de 4 mod 7 es 2, como se ve en la tabla de multiplicar mod 7. Se puede demostrar que un número a solo tiene inverso mod m si es coprimo con respecto a m, es decir, a y m no tienen factores comunes.

2.2 Cifrado de Hill

El cifrado de Hill es un sistema de cifrado de sustitución poligráfica, es decir, en esta clase de cifrado la sustitución del texto claro por texto cifrado se lleva a cabo por bloques, aplicando el algoritmo de sustitución discretamente a cada uno de ellos, y no aplicando una sustitución única a todo el texto, como ocurre en la sustitución monoalfabética, que constituye los sistemas de cifrado más simples, como sería el cifrado César o cifrado por desplazamiento. [2, 4]

El cifrado de Hill suele tener su aplicación a texto, donde cada letra se sustituye por su ordinal en el abecedario y se genera un matriz $n \times n$ de todos los términos módulo z donde z es el número total de caracteres es el alfabeto en cuestión. Sin embargo, nuestra aplicación es a imágenes, ¿y cómo variaría por tanto el procedimiento? Conceptualmente, en nada, ya que lo que hacemos es transcribir las fotos a matrices de los valores de cada píxel, que van de 0 a 255 para cada uno de los tres colores, rojo, verde y azul, y luego proceder sobre estas matrices módulo 256.

En la operación básica de cifrado, el texto a encriptar es un vector columna de n términos módulo z, y la llave de encriptación es una matriz B invertible en $\mathbb{Z}_m$ con lo cual $AB = C$ y el resultado de la operación C es el texto codificado, que es una matriz $n \times n$ módulo z con términos necesariamente distintos de A. De esta forma, para desencriptar C y conseguir el texto claro de nuevo, se procede al camino inverso, es decir A se obtendría multiplicando C por la inversa de B, B^{-1}: $A = CB^{-1}$. Y de hecho el texto a cifrar suele ser una matriz, en nuestro caso es así ya que se trata de una fotografía, por lo tanto, es necesario pasar la matriz a un vector columna de la siguiente forma:

Si lo que se quiere es encriptar una matriz $W = (w_{ij}) \in \mathcal{M}_{n_1 \times n_2}(\mathbb{Z}_m)$ lo que haremos es "transformarla" en un vector columna de longitud $n_1 \times n_2$

$$x = \begin{bmatrix} W_1{}^T \\ W_2{}^T \\ \vdots \\ W_{n_1}{}^T \end{bmatrix} \quad [2.3]$$

donde W_i es el i-ésimo vector fila de W y después proceder como arriba. En el proceso de decodificación una vez obtenido el vector x se reconstruye la matriz W usando

$$w_{ij} = x_{(i-1)n_2+j} \ , \ \ i = 1,2,\dots,n_1 \ , \ \ j = 1,2,\dots,n_2 \quad [2.4]$$

Ya que la operación de encriptado es biyectiva en el sentido en que se obtiene un término de salida por cada término de entrada el vector encriptado es del mismo tamaño que el vector claro y por tanto si recomponemos la matriz esta vez encriptada a partir del vector encriptado, es inmediato que esta es de las mismas dimensiones que la matriz original.

Así sería la codificación básica de un bloque, la unitaria podríamos decir, pero bien, esto es considerando matrices y vectores relativamente pequeños, cuyos cálculos matriciales son relativamente rápidos. Sin embargo, la gran mayoría de las imágenes no se componen de matrices pequeñas, por el contrario, son de tamaños muy grandes y a su vez los vectores columna correspondientes también, luego las matrices de cifrado para el vector íntegro son de dimensiones enormes, y para realizar sobre ellos operaciones de productos y cálculos de inversas se requeriría una potencia de cálculo demasiado grande para los ámbitos de aplicación de este procedimiento. Por lo tanto, la solución pasa por la descomposición del vector de texto claro en vectores más pequeños de tamaños iguales, y luego realizaremos el cifrado 'por bloques', aplicando las operaciones de cifrado a cada uno de estos subvectores. A continuación, desarrollamos este procedimiento teóricamente:

Sea $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}$ y $A = (a_{ij}) \in \mathcal{M}_{N\times N}(\mathbb{Z}_m)$ una matriz cuadrada de orden N con $k \geq N$. Sea l el natural que cumple $(l-1)N < k \leq lN$.

Supongamos que $k = lN$. Sea

$$\tilde{x}_1 = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}, \tilde{x}_2 = \begin{bmatrix} x_{N+1} \\ x_{N+2} \\ \vdots \\ x_{2N} \end{bmatrix}, \dots, \tilde{x}_l = \begin{bmatrix} x_{(l-1)N+1} \\ x_{(l-1)N+2} \\ \vdots \\ x_{lN} \end{bmatrix} \quad [2.5]$$

Se cumple que $x = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \vdots \\ \tilde{x}_l \end{bmatrix}$.

La longitud de cada vector $\tilde{x}_i$ es evidentemente igual al de la matriz A, mientras que x es el vector grande compuesto por todos los $\tilde{x}_i$. En nuestro caso x es el vector que se deriva de la matriz de la foto. Vamos a codificar el vector $\tilde{x}$ a un nuevo vector $\tilde{y}$ de longitud lN construido por bloques mediante:

$$y = \begin{bmatrix} \tilde{y}_1 \\ \tilde{y}_2 \\ \vdots \\ \tilde{y}_l \end{bmatrix} \quad [2.6]$$

donde

$$\tilde{y}_1 = A\tilde{x}_1, \tilde{y}_2 = A\tilde{x}_2, \dots, \tilde{y}_l = A\tilde{x}_l \quad (\text{módulo } m)$$

Todos los productos de matrices con vectores son módulo m. Supongamos que A tienen inversa en $\mathcal{M}_{N\times N}(\mathbb{Z}_m)$. El proceso de decodificación es como sigue: a partir de y obtenemos x

$$\tilde{x}_1 = A^{-1}\tilde{y}_1, \tilde{x}_2 = A^{-1}\tilde{y}_2, \dots, \tilde{x}_l = A^{-1}\tilde{y}_l \quad (\text{módulo } m)$$

Supongamos que $k < lN$. En este caso razonamos de la misma forma para $i = 1,2, \dots, l-1$ y definimos

$$\tilde{x}_l = \begin{bmatrix} x_{(l-1)N+1} \\ x_{(l-1)N+2} \\ \vdots \\ x_k \end{bmatrix} \qquad [2.7]$$

que es un vector de longitud menor que N. Definimos $\tilde{y}_l = A_r\tilde{x}_l \quad (\text{módulo } m)$ siendo $A_{(r)}$ la submatriz principal generada por las $r = k - (l-1)N$ primeras filas y columnas de A, siendo $\tilde{y}_l$ y $\tilde{x}_l$ los últimos subvectores del vector principal que tienen un tamaño de igual o inferior a N.

Supongamos que A y $A_{(r)}$ tienen inversa en $\mathcal{M}_{N\times N}(\mathbb{Z}_m)$ y $\mathcal{M}_{r\times r}(\mathbb{Z}_m)$ respectivamente. El proceso de decodificación sería como antes: a partir de y obtenemos

$$\tilde{x}_1 = A^{-1}\tilde{y}_1, \tilde{x}_2 = A^{-1}\tilde{y}_2, \dots, \tilde{x}_{l-1} = A^{-1}\tilde{y}_{l-1}, \tilde{x}_l = {A_{(r)}}^{-1}\tilde{y}_l \quad (\text{módulo } m)$$

siendo A^{-1} y ${A_{(r)}}^{-1}$ las inversas de A y $A_{(r)}$ modulo m.

El vector decodificado original sería:

$$x = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \vdots \\ \tilde{x}_l \end{bmatrix} \qquad [2.8]$$

También es posible sustituir la matriz $A_{(r)}$ por cualquier otra matriz de orden r no singular (modulo m). Una solución inicial que se concibió para la existencia de términos restantes en el vector principal, es decir, suponiendo que $k < lN$, era rellenando con ceros hasta que el vector tenía el tamaño lN. Parecía una solución simple y efectiva a priori, pero no era del todo correcta, ya que en el último subvector, el $\tilde{x}_l$, a causa de los ceros rellenados, en la operación de cifrado los ceros se multiplican por términos de la matriz de cifrado dando ceros que no influyen en el resultado, pero al descifrar, multiplicamos la matriz de descifrado por los vectores cifrados que venían del cálculo con ceros, y no se vuelven a obtener los ceros con una operación multiplicativa, luego el vector descifrado que obtenemos no es igual al texto claro inicial. Por tanto, esta operación se tiene que realizar con una matriz de cifrado reducida $A_{(r)}$ que se ajuste al tamaño de este subvector.

Dicho esto, el procedimiento anterior presenta el inconveniente de que si alguno de los vectores $\tilde{x}_j$ ($j \in \{1,2, \dots, l\}$) es el vector nulo entonces lo mismo le ocurre a $\tilde{y}_j$ y en ese caso no se ha producido encriptación. Una posible forma de evitar esto es utilizar además un vector v no nulo de longitud N para la codificación

$$\tilde{y}_j = A\tilde{x}_j + v \quad (\text{módulo } m) \qquad [2.9]$$

La decodificación sería entonces

$$\tilde{x}_j = A^{-1}\left(\tilde{y}_j - v\right) \ (\text{módulo } m) \qquad [2.10]$$

Por tanto, para hacer un procedimiento todavía más robusto, pasaremos de un cifrado de Hill únicamente con producto a un cifrado con combinación lineal, a uno con producto matricial y suma de vectores. Suponiendo que se trabaja con múltiples matrices $(A_1, A_2, \dots, A_l)$ y vectores $(v_1, v_2, \dots, v_l)$:

$$\text{codificación:} \qquad \tilde{y}_j = A_j\tilde{x}_j + v_j \ (\text{módulo } m) \qquad [2.11]$$

$$\text{decodificación:} \qquad \tilde{x}_j = A_j^{-1}\left(\tilde{y}_j - v_j\right) \ (\text{módulo } m) \qquad [2.12]$$

Es importante recordar que, dado que las matrices A_j deben tener una inversa, es decir, tienen que ser no singulares, su determinante tiene que ser no nulo, pero, además, ya que estaremos trabajando en módulo m, la otra condición es que el determinante de la matriz en cuestión sea coprimo con respecto al módulo m.

Brevemente mostraremos cómo se implementará una ejecución del algoritmo de cifrado de Hill entero en Matlab, donde separaremos las imágenes en tres matrices de rojos (R), verdes (G) y azules (B):

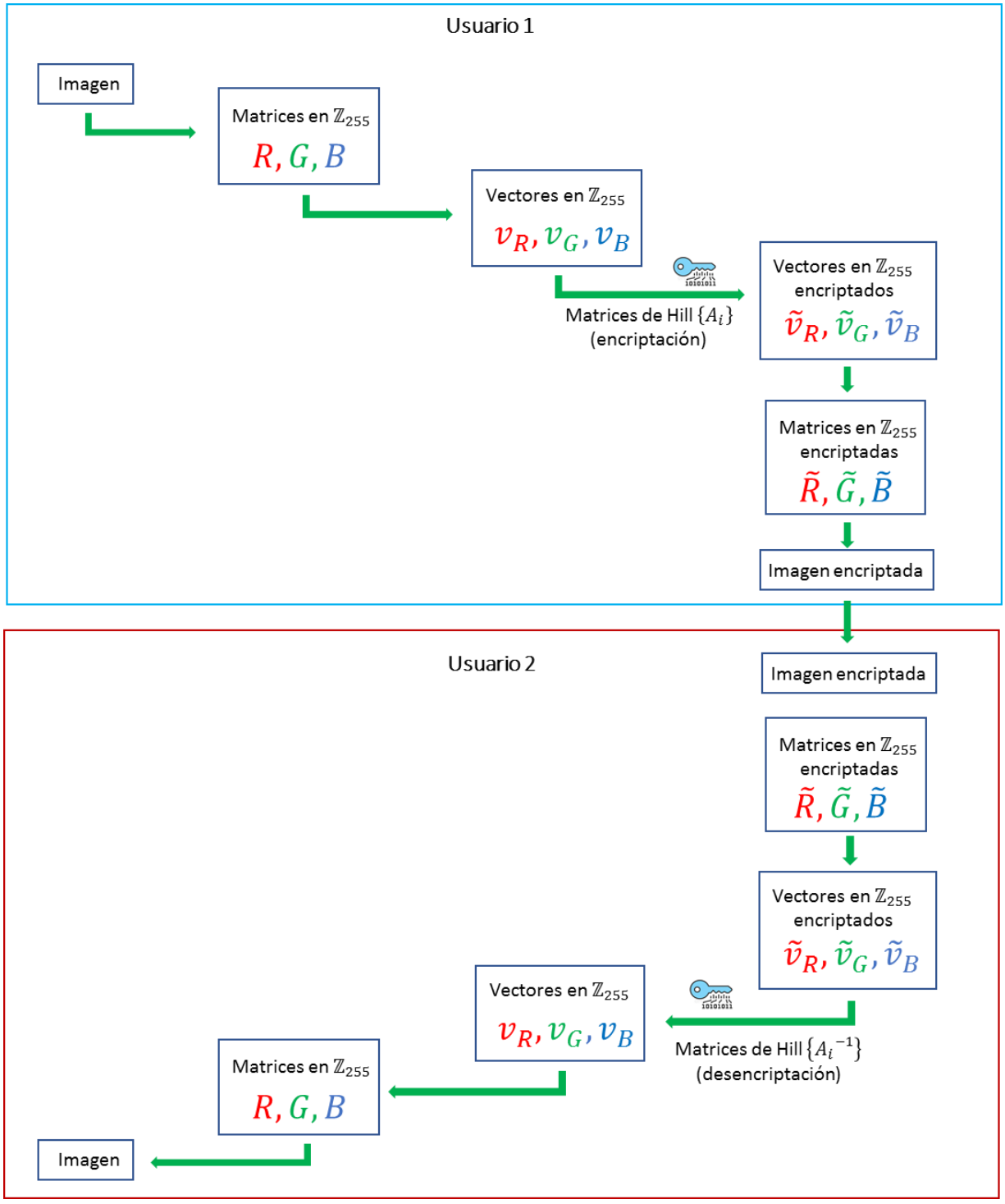

Figura 4. Esquema del cifrado de Hill implementado a Matlab

2.3 Números aleatorios y fractales

2.3.1 Generación de números aleatorios

Nuestro sistema de cifrado requiere de un generador de números aleatorios, pero estos ocurren verdaderamente sólo en la naturaleza, ya que son intrínsecos de los sistemas analógicos. Ejemplos de auténtica aleatoriedad son el ruido blanco, los resultados del lanzamiento de monedas, fichas o dados, o la caída libre de canicas por secciones con obstáculos periódicos (máquina de Galton no centrada), y si buscamos números aleatorios habríamos de hacer ensayos de estos sucesos e introducirlos al ordenador. Sin embargo, queremos un generador que podamos configurar con datos iniciales, por tanto, recurrimos a algoritmos computacionales programables que ejecutamos en el ordenador. Pero estos no son verdaderamente aleatorios, ya que consisten en una serie de reglas finitas, lo que los hace determinísticos, es decir, sus resultados son predecibles, además de que los estados que el procesador digital puede adoptar son discretos, por lo tanto, llegados a un cierto número de iteraciones, habrá periodicidad en los resultados. Estos números por tanto no son aleatorios sino pseudoaleatorios y el algoritmo que los genera es igualmente un generador pseudoaleatorio. [9]

2.3.2 Fractales

Según el diccionario de la Real Academia Española de la Lengua, desde el punto de vista matemático, un fractal es una "estructura iterativa que tiene la propiedad de que su aspecto y distribución estadística no cambian cualquiera que sea la escala con que se observe." El término "fractal" fue acuñado por el matemático francés Benoit Mandelbrot y proviene de la palabra latina *fractus* que se traduce como quebrado, roto, fracturado, fragmentado. Geométricamente es un objeto que tiene la propiedad de que se va repitiendo el mismo patrón a diferentes escalas y orientaciones. [10]

Los fractales presentan las siguientes características:

- *Simetría de escala* de forma que su estructura básica, fragmentada o irregular, se repite a diversas escalas
- *Autosimilitud,* su aspecto no depende del observador, y sucesivas ampliaciones reproducen la imagen original.
- *Recursividad,* que es la cualidad de estar definido dentro de sí mismo.
- *Infinitud,* ya que es infinitamente reproducible tras sucesivas ampliaciones.
- *Irregularidad* e imposibilidad de ser descritos geométricamente de forma tradicional
- *Dimensión no entera,* al tener una estructura interna repetitiva no son definibles en un espacio *n*-dimensional siendo *n* un número entero.

Los primeros fractales matemáticos se construyeron a partir de la iteración de funciones. Uno de los primeros ejemplos son los conjuntos de Julia, generados a partir del método iterativo

$$z_{n+1} = {z_n}^2 + c, \qquad n = 0,1,2,\ldots \qquad [2.13]$$

donde c es un número complejo dado. Para cada punto de un rectángulo del plano complejo z_0 le asignamos un color que dependa de la velocidad con que la sucesión de iterados $\{z_n\}$converja o diverja a ∞. Para algunos valores de c (por ejemplo $c = 0.36 + 0.1i$) se genera una figura que resulta ser un fractal (Figura 5). Con más generalidad pueden generarse fácilmente fractales a partir de métodos iterativos:

$$z_{n+1} = h(z_n), \qquad n = 0,1,2,\ldots \qquad [2.14]$$

donde h: $\mathbb{C} \longrightarrow \mathbb{C}$ es una función compleja. Otro ejemplo puede verse en la Figura 6.

La idea es usar las premisas de estos generadores de fractales en nuestro generador de números pseudoaleatorios, para generarlos gracias a su multiplicidad. Sin embargo, hay que recalcar que existen muchos otros métodos para generar figuras fractales.

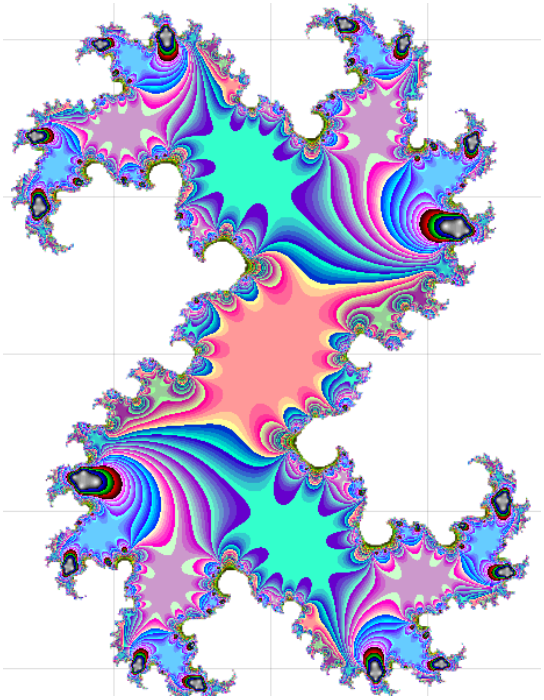

Figura 5. Dragón de Mandelbrot

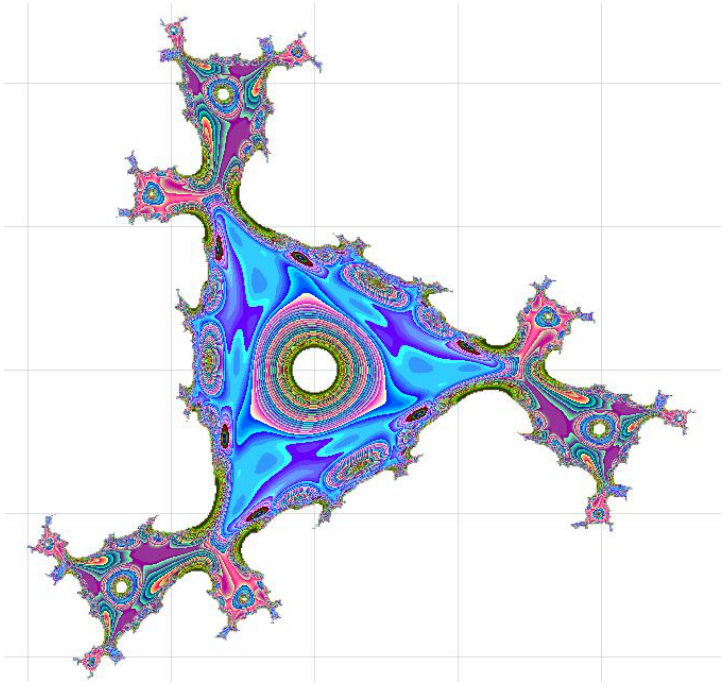

Figura 6. Conjunto de Julia correspondiente a $h(z) = z^3 - 0.85\sqrt{|z|} + 0.6i$

2.4 Generación de matrices pseudoaleatorias

La parte fundamental que implementamos para otorgar resistencia al algoritmo de cifrado de Hill es el cálculo con matrices pseudoaleatorias, vamos a generar matrices cuadradas de números pseudoaleatorios utilizando la misma técnica que permite generar fractales. Pero si realmente sólo necesitamos matrices compuestas de números aleatorios, ¿por qué no definir números aleatorios y escribir una matriz de ellos? Es más, ¿por qué no utilizar la función de Matlab que permite generar matrices aleatorias de cierto tamaño? Pues simplemente porque necesitamos que las matrices generadas sean trazables a partir de unas variables iniciales, si las matrices pueden ser reproducibles con unas variables en particular, entonces podemos usar esos valores como clave para reproducirlas y así obtener sus inversas, ya que serán no singulares, y así implementar el cifrado de Hill.

Supongamos que queremos generar una matriz pseudoaleatoria de tamaño $N \times N$ donde sus entradas sean valores en $\mathbb{Z}_m = \{0,1,2,\dots,m-1\}$. Consideremos una función en el plano complejo $\mathbb{C} \equiv \mathbb{R} \times \mathbb{R}$:

$$h\colon [a,b] \times [c,d] \to \mathbb{C} \qquad [2.15]$$

y la función $\varphi\colon \mathbb{C} \to [a,b] \times [c,d]$ dada por

$$\varphi(z) = re(z_1) - (b-a)E\left[\frac{re(z_1)-a}{b-a}\right] + i\left(im(z_1) - (d-c)E\left[\frac{im(z_1)-c}{d-c}\right]\right) \qquad [2.16]$$

donde $re(\cdot)$ es la parte real, $im(\cdot)$ es la parte imaginaria y $E[\cdot]$ es la parte entera. Llamamos $H = \varphi \circ h$. Claramente $H([a,b] \times [c,d]) \subset [a,b] \times [c,d]$.

Dado un natural N dividimos los intervalos $[a,b]$ y $[c,d]$ en $N-1$ subintervalos y llamemos

$$x_r = a + (r-1)\frac{b-a}{N-1}, \qquad r = 1,2,\dots,N \qquad [2.17]$$

$$y_s = c + (s-1)\frac{d-c}{N-1}, \qquad s = 1,2,\dots,N \qquad [2.18]$$

Geométricamente hemos generado un mallado $N \times N$ en el rectángulo del plano complejo $[a,b] \times [c,d]$.

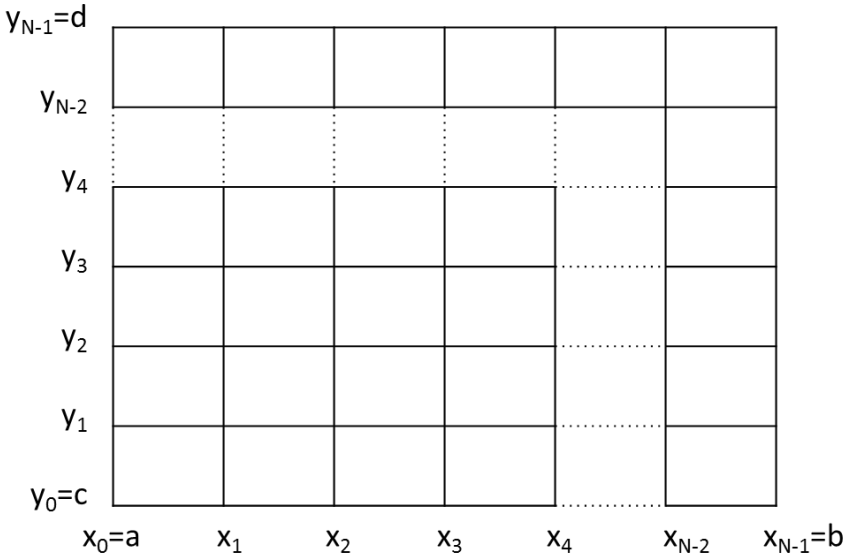

Figura 7. Mallado N × N del intervalo [a,b,c,d]

Sea:

$$z_{rs}^{(0)} = z_0 \coloneqq x_r + iy_s, \qquad r = 1,2,\dots,N, s = 1,2,\dots,N \tag{2.19}$$

Sea $z_0 = x_r + iy_s$, generamos $z_{rs}^{(1)} = z_1 = h(z_0)$. Está claro que z_1 no tiene por qué estar en el intervalo $[a,b] \times [c,d]$, pero podemos "llevarlo" a ese rectángulo del plano complejo de la siguiente forma:

$$\tilde{z}_{1r} = a + re(z_1) - (b-a)E\left[\frac{re(z_1) - a}{b-a}\right] \in [a,b] \tag{2.20}$$

$$\tilde{z}_{1s} = c + im(z_1) - (d-c)E\left[\frac{im(z_1) - c}{d-c}\right] \in [c,d] \tag{2.21}$$

donde $re(\cdot)$ es la parte real, $im(\cdot)$ es la parte imaginaria y $E[\cdot]$ es la parte entera.

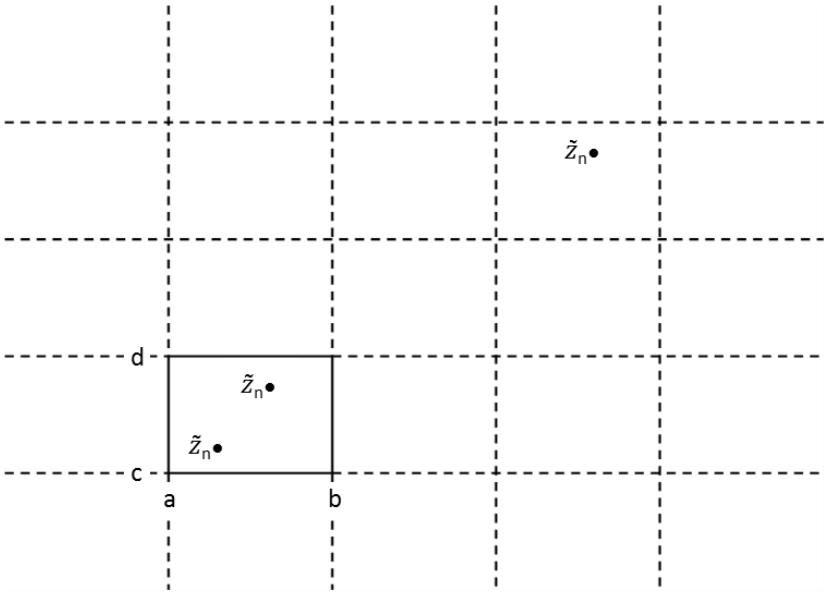

Figura 8. Secuencia de generación de números pseudoaleatorios en el mallado

El número complejo $\tilde{z}_1 = \tilde{z}_{1r} + i\tilde{z}_{1s}$ se encuentra en el rectángulo inicial. Utilizando la función H podemos generar la sucesión de iterados $\{\tilde{z}_k\}$ definida por

$$\tilde{z}_k = H(\tilde{z}_{k-1}), \quad k = 1,2,\dots \tag{2.22}$$

Como $H = \varphi \circ h$, generamos $z_2 = h(\tilde{z}_1)$, y a continuación aplicamos φ, es decir: $\tilde{z}_2 = \varphi(z_2)$, con el que, como antes, "llevamos" este valor al rectángulo inicial obteniendo $\tilde{z}_2$ y vamos realizando este procedimiento n veces.

Fijando ahora dos naturales n y q sean:

$$\alpha_{rs} = 10^q re(\tilde{z}_n) - E[10^q re(\tilde{z}_n)] \tag{2.23}$$

$$\beta_{rs} = 10^q im(\tilde{z}_n) - E[10^q im(\tilde{z}_n)] \tag{2.24}$$

Por construcción $\alpha_{rs}, \beta_{rs} \in [0,1]$. Finalmente podemos definir

$$a_{rs} = E[m\alpha_{rs}], \qquad b_{rs} = E[m\beta_{rs}] \tag{2.25}$$

Hemos generado dos matrices pseudoaleatorias en $\mathbb{Z}_m$

$$A = (a_{rs}), \quad B = (b_{rs}) \qquad [2.26]$$

Estas matrices son, por tanto, las que emplearemos como matrices de cifrado en el procedimiento de cifrado de Hill, donde si la operación de codificación y decodificación es:

$$\text{codificación:} \qquad \tilde{y}_j = A_j \tilde{x}_j + v_j \quad (\text{módulo } m) \qquad [2.11]$$

$$\text{decodificación:} \qquad \tilde{x}_j = A_j{}^{-1}(\tilde{y}_1 - v) \quad (\text{módulo } m) \qquad [2.12]$$

la matriz A_j la podemos tomar indistintamente de A o B al ser ambas de igual tamaño y necesariamente pseudoaleatorias e independientes, y v_j se puede obtener de cualquier columna de A o B indistintamente.

2.5 Codificación asimétrica y algoritmo de Diffie-Hellman

2.5.1 Codificación asimétrica o en clave pública

Hasta ahora, el sistema de codificación planteado es un cifrado muy básico, pero robusto. Básico en el sentido en que responde a los sistemas típicos de codificación en cuanto a que el texto claro se cifra con una clave, se envía encriptado y el receptor, que recibe texto codificado y la clave y decodifica el texto con la misma clave con que se codificó. Y robusto, porque el algoritmo de generación de matrices pseudoaleatorias ofrece a priori una tremenda variabilidad e independencia de los resultados, lo que se traduce en una dificultad enorme para romper la clave desde fuera por 'fuerza bruta', es decir, por procedimientos iterativos de prueba y error de valores. Hay demasiadas combinaciones de valores posibles que hacer para descifrar incluso una fotografía pequeña.

Hemos creado un sistema de codificación bidireccional (ya que la información fluye en ambos sentidos) en clave privada o codificación simétrica, donde la clave utilizada para cifrar es la misma que para descifrar. Luego la clave es única, y su seguridad es limitada, ya que si un supuesto 'hacker' externo intercepta la clave y conoce el algoritmo de cifrado puede romper la encriptación.

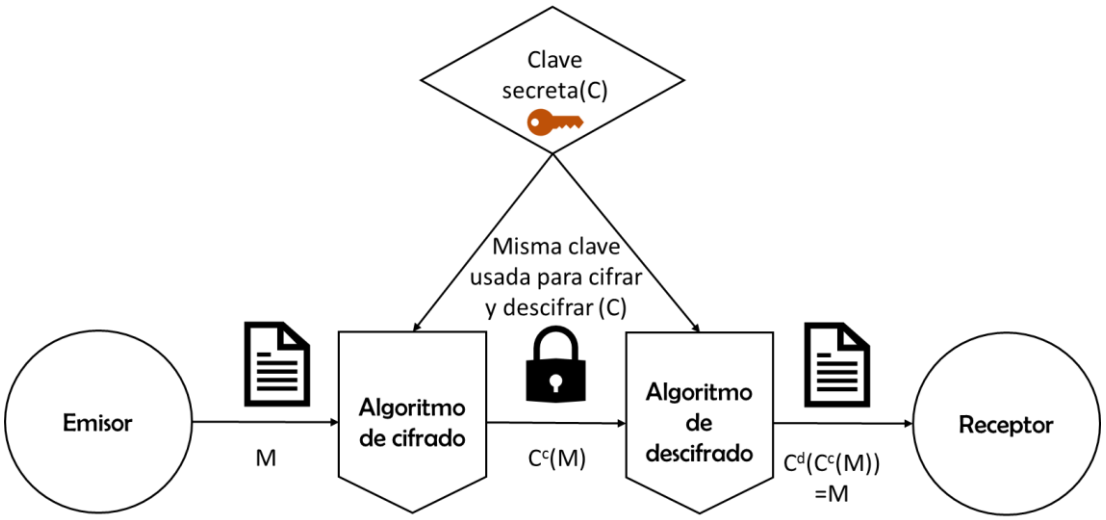

Figura 9. Codificación simétrica

Por otro lado, existe el sistema de cifrado bidireccional asimétrico, en el cual existen dos claves asociadas: la pública, que es la que se usa para cifrar el mensaje, y la privada, que es únicamente conocida por los respectivos participantes en el cifrado y no se envía en ningún momento. Lo que es interesante con este sistema de cifrado es que lo que se cifra con la clave pública se descifra con la privada, lo cual parece contraintuitivo, pero es así, e implica que, en caso de intercepción por un externo, es imposible descifrar el mensaje, suponiendo que no conoce la clave privada, que sería el caso en condiciones normales. Ofreciendo esta última capa de seguridad que harán del sistema increíblemente fiable. [3]

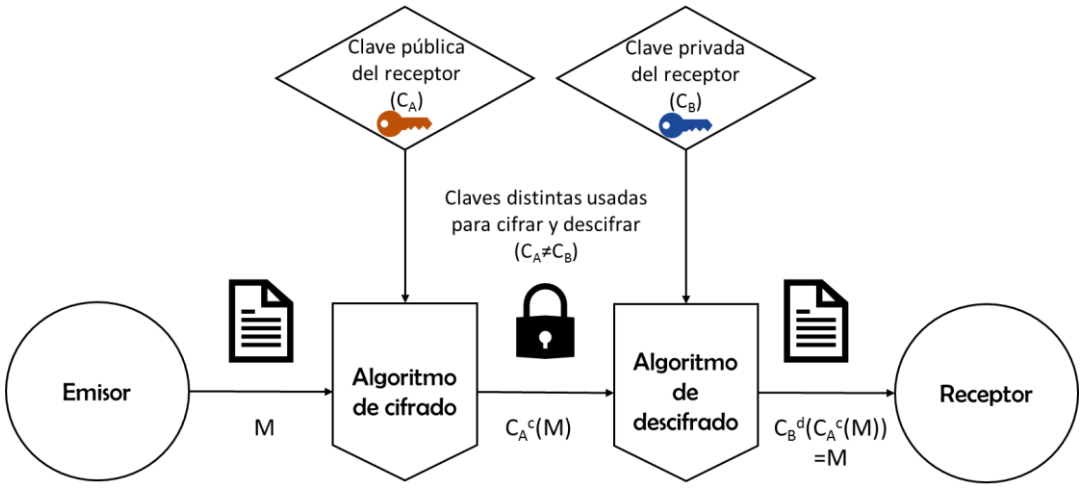

Figura 10. Codificación asimétrica

2.5.2 Algoritmo de Diffie-Hellman

Existen varios algoritmos de cifrado asimétrico distintos, los principales ejemplos son: el RSA, ElGamal o el Rivest-Shamir-Adleman (RSA) que son los nombres de sus creadores, que se basa en el cálculo de productos de números primos muy grandes, que son muy difíciles de factorizar en aritmética modular, o el algoritmo Diffie-Hellman, propuesto por Whitfield Diffie y Martin Hellman en 1976, este último es el que vamos a utilizar en nuestro cifrado para generar claves públicas y privadas que complementen el algoritmo base de cifrado de Hill. [2,5]

El algoritmo Diffie-Hellman basa su resistencia a decodificaciones externas en la dificultad computacional de calcular logaritmos discretos en números primos muy grandes. El funcionamiento es el siguiente:

Definimos dos participantes, les llamaremos *Antonio* y *Laura* por simplicidad.

- Indistintamente uno de ellos, preferiblemente el que vaya a enviar primero, define un primo m y un generador $g \in \mathbb{Z}_m$.

- A continuación, *Antonio* y *Laura* escogen sus respectivos términos privados, a y b, siendo $\{a, b\} \in \mathbb{Z}_{m-1}$, y calculan sus respectivas claves públicas, *Antonio* calcula $A = g^a \ mod \ m$ y *Laura* calcula $B = g^b mod \ m$
- Ahora, y aquí está la esencia del algoritmo, tanto *Antonio* como *Laura* pueden calcular su clave secreta compartida, K, con $K = g^{a \cdot b} \ mod \ m$. De acuerdo con las propiedades del grupo $\mathbb{Z}_m$ Antonio calcularía:

$$B^a = (g^b mod\ m)^a mod\ m = \overbrace{\big((g^b mod\ m) \cdot (g^b mod\ m) \dots (g^b mod\ m)\big)}^{a \text{ veces}} mod\ m$$
$$= g^{a \cdot b} mod\ m = K \qquad [2.27]$$

- De la misma forma, el cálculo de *Laura* sería simétrico a este:

$$A^b = (g^a mod\ m)^b mod\ m = \overbrace{((g^a mod\ m) \cdot (g^a mod\ m) \dots (g^a mod\ m))}^{b \text{ veces}} mod\ m$$
$$= g^{b \cdot a} mod\ m = K \qquad [2.28]$$

Por tanto, ya que ambos participantes pueden calcular K desde distintos caminos, esta será la clave común que ambo utilizan realmente para cifrar y descifrar, pero enviarán A y B.

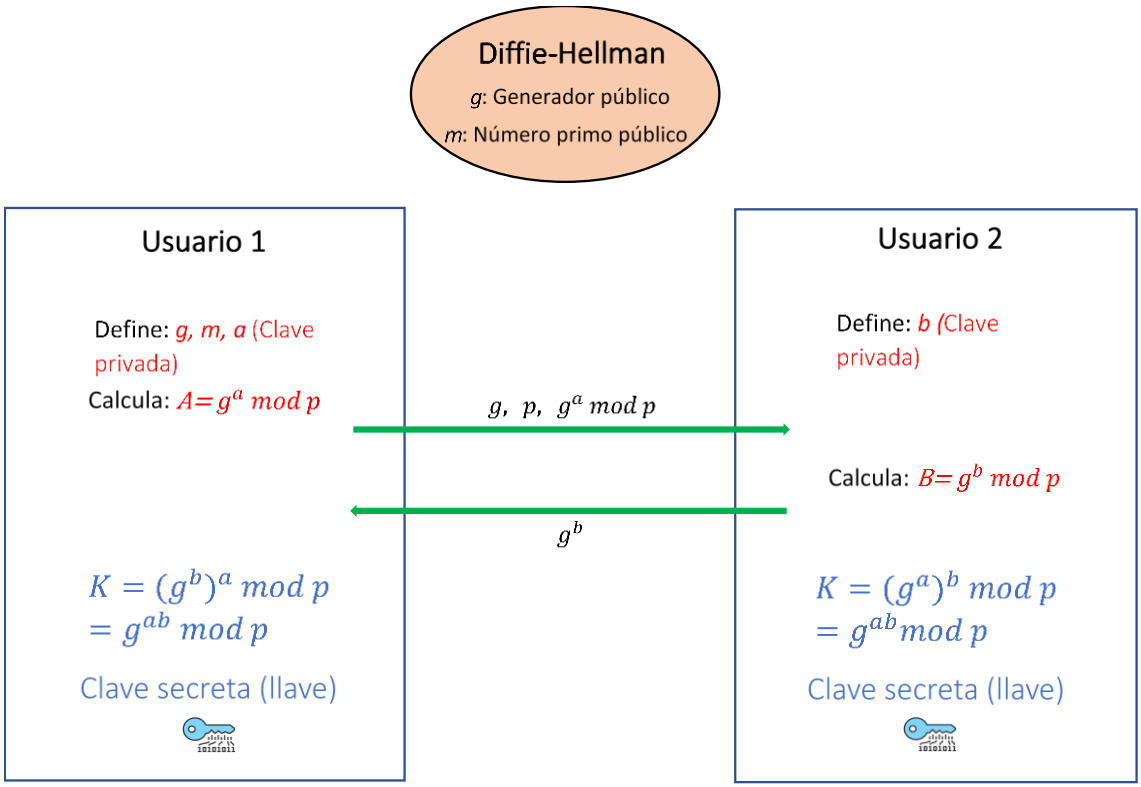

Figura 11. Algoritmo de Diffie-Hellman

La fortaleza del algoritmo frente a ataques de fuerza bruta proviene de la dificultad del cálculo del algoritmo en módulo *m.* Suponiendo que un adversario o 'hacker' exterior conoce g, m, A y B, ya que los puede interceptar en el envío. Necesitaría obtener a o b para poder calcular K, que al final es la clave secreta compartida de descifrado, pero obtener a o b desde A y B significa invertir la función exponencial y calcular el logaritmo discreto:

$$a = \log disc_g A \quad \text{y} \quad b = \log disc_g B \tag{2.29}$$

Lo cual es computacionalmente extremadamente difícil con números lo bastante grandes, debido a la multiplicidad de soluciones. Por ejemplo, si $5^c \ mod\ 11 = 3$, una solución es $c = 2$ ya que $5^2 \ mod\ 11 = 25 \ mod\ 11 = 3$. Pero esta no es la única solución, ya que: $5^9 mod\ 11 = 1$, con lo que: $5^{2+9n} mod\ 11 = 1$, por lo tanto, la ecuación tiene infinitas soluciones $c = 2 + 9n, n \in \mathbb{N}$, no es inmediato definir la correcta.

Sin embargo, cómo se ha visto antes, el cálculo de la exponencial en mod m es un cálculo inmediato, con lo que tenemos el cifrado calculable en una dirección, pero el descifrado desde fuera computacionalmente inviable.

Como se ha comentado, la fortaleza del algoritmo radica en el uso de números primos muy grandes, ya que al calcular sus exponentes se sacan números muy grandes muy difíciles de factorizar por un hacker. Sin embargo, Matlab no está programado para trabajar con números tan grandes y muestra problemas de desbordamiento al calcular exponentes del orden de dos dígitos y bases de más de 10, esto reduce significativamente el rango de valores que podemos tomar para encriptar nuestra llave y por tanto facilita enormemente el trabajo del hacker. Por tanto, la solución ingeniosa que hemos tomado a esta situación es la de diversificar las llaves, es decir, usar vectores para los valores de a y b, de tamaño p y q respectivamente, que a su vez resultan en que A y B son vectores también:

$$a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_q \end{bmatrix} \quad \text{y} \quad A = \begin{bmatrix} g^{a_1} \\ g^{a_2} \\ \vdots \\ g^{a_p} \end{bmatrix}, \quad B = \begin{bmatrix} g^{b_1} \\ g^{b_2} \\ \vdots \\ g^{b_p} \end{bmatrix} \tag{2.30}$$

Mientras que la clave secreta compartida K será una matriz de la forma:

$$K = \begin{pmatrix} g^{a_1 \cdot b_1} & g^{a_1 \cdot b_2} & \cdots & g^{a_1 \cdot b_q} \\ g^{a_2 \cdot b_1} & g^{a_2 \cdot b_2} & & g^{a_2 \cdot b_q} \\ \vdots & & \ddots & \vdots \\ g^{a_p \cdot b_1} & g^{a_p \cdot b_2} & \cdots & g^{a_p \cdot b_q} \end{pmatrix} \tag{2.31}$$

3. Desarrollo de la programación

3.1 Aplicación a la encriptación digital de imágenes

Tras haber desarrollado todos los conceptos teóricos detrás del funcionamiento de nuestra programación, vamos a discutir cómo se lleva a cabo esta implementación de los algoritmos a Matlab.

En primer lugar, el tratamiento de las imágenes lo realizaremos enteramente con Matlab. Con el comando `imread` Matlab 'lee' una imagen y la traduce a un *array* tridimensional compuesto de tres matrices correspondientes a los colores rojo, verde y azul de la imagen, que son los colores fundamentales utilizados en imágenes digitales para componer todas las tonalidades mostradas por una pantalla, los términos de estas matrices son escalares con rango de 0 a 255 que representan la intensidad de cada color en el píxel que ocupa una posición en la imagen igual a la del término en la matriz, donde un valor de 255 significa intensidad máxima de ese color en el píxel en cuestión y de 0 ninguna presencia del color; el color negro equivale a un valor de 0 de esa posición en las tres matrices y el blanco a un valor de 255 en las tres matrices.

La programación de los algoritmos se ha llevado a cabo en 10 programas fundamentales y 3 programas auxiliares. Los principales son: `codefoto1`, `codefoto2`, `codefoto3`, `decodefoto1`, `decodefoto2`, `decodefoto3`, `Usuario1a_DH`, `Usuario1b_DH`, `Usuario2a_DH`, `Usuario2b_DH`; mientras que: `invmod`, `invm` y `genemat` complementan a los principales.

`Codefoto1` es el algoritmo principal de codificación fotográfica, donde transponemos las imágenes a vectores, descomponemos estos y los ciframos por cifrados de Hill, iterando para obtener matrices pseudoaleatorias nuevas en cada operación con cada sub-vector. `Codefoto2` realiza el mismo algoritmo con la diferencia de que no calcula nuevas matrices para cada nueva operación, sino que itera una sola vez y genera una sola matriz que es la que se utiliza para todos los cálculos. C`odefoto3` sigue el mismo procedimiento que `codefoto2` usando una sola matriz, pero además se aplica previamente un proceso adicional que lleva a cabo un intercambio de píxeles en la foto (usando técnicas de la aritmética modular). `Decodefoto1`, `decodefoto2`, `decodefoto3` son por tanto los programas de desencriptación respectivos, que recuperan la foto clara a partir de la codificada con la clave correcta. `Genemat` es el programa que genera las matrices aleatorias a partir de una serie de parámetros, y es llamado dentro de los programas de codificación, y también de los de decodificación para calcular las inversas. Por otro lado, `invm` calcula el inverso de un valor módulo m, y `invmod` se encarga de calcular la inversa de una matriz en módulo m.

A continuación, tenemos los programas del algoritmo de Diffie-Hellman. `Usuario1a_DH` es el programa que calcula la clave pública del primer usuario o el emisor. `Usuario2a_DH` calcula de clave pública del segundo usuario o el receptor. `Usuario1b_DH` y `Usuario2b_DH` calculan respectivamente la clave secreta compartida para ambos usuarios.

Además, Matlab nos ofrece multitud de programas y comandos que utilizaremos en nuestra programación. En primer lugar, la función `mod(a,b)` proporciona el valor de `a` módulo `b`. La función `gcd(a,b)` calcula el máximo común divisor entre `a` y `b`. La función `round(a)` redondea el valor `a` al entero más cercano. La función `flood(a)` trunca el valor `a`, es decir, lo redondea al entero menor. Además, usaremos comandos que nos permiten exponer la información de nuestros resultados, como la función `tic toc` que nos permitirá cronometrar el tiempo de ejecución de los programas para evaluar su rendimiento más adelante. La función `histogram(X,y)` genera un histograma de `y` grupos de la distribución `X`.

3.2 Programación de Invm e Invmod

3.2.1 Programa Invm

Al ejecutar `invm` obtenemos el inverso `y` de un número `x` en mód `m`.

```
function y = invm(x,m)
y = [];
for i=1:m-1             % Recorremos de 0 a m
    if mod(x*i,m)==1    % Si el producto de x y el contador en este
      % instante es igual a 1 mod m, hemos encontrado el inverso y paramos
        y = i;
    end
end
```

3.2.2 Programa Invmod

`Invmod` es la función programada para calcular la matriz inversa `B` de una cierta matriz `A` en mod m, siendo esta matriz `A` y `m` sus entradas. `invmod` llama a `invm` para calcular el inverso del determinante de la matriz que invertimos.

```
function B = invmod(A,m)
y = round(mod(det(A),m));     % Calculamos el determinante de A mod m y
                  % lo redondeamos para evitar errores de redondeo
if (y==0)|(gcd(y,m)==2)       % Doble condición de no singularidad
    B =[];

else
    C = A;
    [n1,n2] = size(A);        % Guardamos las dimensiones de A, n1 y n2
    for i=1:n1
        for j=1:n2
            C(i,j) = (-1)^(i+j)*det(A([1:i-1,i+1:n1],[1:j-1,j+1:n2]));
                              % Calculamos la adjunta de la matriz A
        end
    end
    C = round(mod(C,m));  % Calculamos C mod m y redondemaos los términos
    z = invm(y,m);   %Calculamos el inverso del determinante de A, y, mod m
    B = mod(C'*z,m); %Multiplicamos el inverso del determinante por
                      %la traspuesta de C, obteniendo la inversa
end
```

3.3 Programación de Codefoto1

A continuación, se expone el código del programa `codefoto1` comentado. `codefoto1` entra con las variables: `foto`, la foto que queremos codificar. `cfoto`, el archivo que contendrá la foto una vez codificada. `h`, la función en términos de z y t con la que iteraremos la generación de matrices pseudoaleatorias. `I`, consta de cuatro valores y es el rectángulo en $\mathbb{C}$ donde realizamos el mallado y trasladamos el valor de entrada de la función en cada iteración. `N`, que es el orden que tendrán las matrices cuadradas pseudoaleatorias, `m`, el mod del espacio $\mathbb{Z}_m$ donde trabajaremos con el cifrado de Hill. `n`, el número de iteraciones para generar una matriz pseudoaleatoria. `q`, es el número de cifras decimales que trasladamos para obtener los valores aleatorios finales. `t0` y `dt` definen a la variable t en la función `h`, que es sobre la que varía en cada iteración para proporcionar variabilidad a los números aleatorios generados, `t0` es el valor inicial de t y `dt` es el incremento.

Estas variables son lo que se utilizan como llave de encriptación, el usuario emisor, tras ejecutar `codefoto1` envía la foto encriptada junto con estos valores al receptor, que usa la clave para recorrer el camino inverso y recuperar la foto original. Existen algunas restricciones a los valores que pueden tomar las variables de la clave, en primer lugar, `I`, ha de tener el primer y tercer valor menores que el segundo y cuarto, ya que definen un rectángulo en el espacio, `m` ha de ser un menor o igual a 255, y, además, `q` no ha de tomar más grandes que 14, ya que cuando se desplazan tantas posiciones decimales en los números se incurre en posible error de redondeo por parte de las operaciones internas a Matlab en las cifras menos significativas de los valores.

Estas variables son las mismas que se utilizan en los siguientes programas derivados de `codefoto1`.

```
function [XNR,XNG,XNB]=codefoto1(foto,cfoto,h,I,N,m,n,q,t0,dt)
% foto nombre de la foto a codificar
% cfoto nombre de la foto codificada
% h expresión de texto con las letras z y t que recoge la función iterativa
% I = [a,b]x[c,d] intervalo de definición de la h
% N orden de las matrices aleatorias (A1 y A2)
% m módulo(Z_m)
% n número de iteraciones
% q posición a partir de la cual se cuentan los decimales
% t0 valor inicial de la variable t
% dt incremento de la variable t
% XR, XG y XB son matrices en Z_256 correspondientes a los planos en escala
%de rojos,verdes y azules respectivamente de la fotografía a codificar
% XNR, XNG y XNB son matrices en Z_256 correspondientes a los planos en
%escala de rojos, verdes y azules respectivamente de la fotografía
%codificada
tic  % Iniciamos cronometración
f = imread(foto); % Asignamos el array tridimensional de colores de la foto
                  % a la variable f
X = double(f);  % Pasamos f a formato double
      % Definimos las matrices XR, XG y XB cómo las matrices de los valores
      % de cada color, que son las tres matrices que se descomponen del
      % array en posición 1,2 y 3
XR = X(:,:,1);
XG = X(:,:,2);
XB = X(:,:,3);
```

```
[n1,n2] = size(XR);      % Definimos n1 y n2 cómo las dimensiones horizontal
                         % y vertical de las matrices X (no hace falta decir
                         % que las tres tienen la misma)
vr  = [];% Creamos tres vectores vacíos que reconfiguraremos posteriormente
vg  = [];
vb  = [];
      % Pasar XR,XG,XB (matriz) a vr,vg,vb (vector columna)
      % Recorremos las matrices fila a fila y colocamos los términos
      % consecutivamente en columna en los vectores que hemos creado antes,
      % generando vectores columna paracada matriz
for i=1:n1
    for j=1:n2
        vr = [vr;XR(i,j)];
        vg = [vg;XG(i,j)];
        vb = [vb;XB(i,j)];
    end
end
nn = n1*n2;  % Definimos nn como el número de términos de las matrices
k = floor(nn/N);  % Definimos k como el número de vectores de tamaño N que
                  % cabrían perfectamente en un vector de tamaño nn
r = mod(nn,N);    % Definimos r como el número de términos sobrantes del
                  % cociente entre nn y N
                  % Trasladamos los vectores v a los vectores u
ur = vr;
ug = vg;
ub = vb;
s = 0;            % Generamos s
t = t0;           % 't' toma el valor inicial de este ('t0')
if r>0      % En primer caso tratamos los términos restantes del
            % vector columna que no entran en la multiplicación por
            % matrices de tamaño N
    [A_1,A_2] = genemat(h,I,N,m,n,q,t);% Llamamos a la función genemat
                                % para generar las matrices pseudoaleatorias
    y = round(mod(det(A_1(1:r,1:r)),256));     % Calculamos 'y' cómo el
            % determinante de la submatriz (de tamaño rxr) de la matriz A_1
            % generada por genemat módulo 256
    while (y==0)||(gcd(y,2)==2)  % Condición de que la matriz A_1 sea
      % singular: 'y'=0 o 'y' y 2 (mínimo factor de 256) no sean coprimos
        t = t+dt;  % Aumentamos 't'
        [A_1,A_2] = genemat(h,I,r,m,n,q,t);  % Recalculamos matrices para
      % cada iteración con un 't' nuevo
        y = round(mod(det(A_1(1:r,1:r)),256));   % Calculamos de nuevo 'y'
      % para cada matriz para seguir asegurando si inversa
    end
    Ar1 = A_1(1:r,1:r); % Calculamos las versiones reducidas a los rxr
      % primeros términos de A1 y A2
    Ar2 = A_2(1:r,1:r);
      % Generamos usando un cifrado de Hill modificado las últimas 'r'
      % entradas encriptadas
    ur(1+N*k:nn) = mod(Ar1*(vr(1+N*k:nn)+Ar2(:,1)),256);  % Completamos los
      % términos restantes, en caso de haberlos, con el producto con las
      % matrices reducidas
    ug(1+N*k:nn) = mod(Ar1*(vg(1+N*k:nn)+Ar2(:,1)),256);
    ub(1+N*k:nn) = mod(Ar1*(vb(1+N*k:nn)+Ar2(:,1)),256);
end
while s<k % Generamos usando un cifrado de Hill modificado con una sucesión
      % dinámica de matrices pseudoaleatorias las entradas encriptadas
    t = t+dt;  % Aumentamos 't'
    [A1,A2] = genemat(h,I,N,m,n,q,t);  % Generamos matrices
```

```
    y = round(mod(det(A1),256));  % Calculamos 'y'cómo el determinante de
      % la submatriz (de tamaño NxN) de la matriz A_1 generada por genemat
      % módulo 256
    if (y~=0)&&(gcd(y,2)==1)      % Condición: A1 es singular
        s = s+1;                  % Codificación
        ur(1+N*(s-1):s*N) = mod(A1*(vr(1+N*(s-1):s*N)+A2(:,1)),256);
  % Definimos el valor de los términos de los 'u's en los sucesivos
  % intervalos del tamaño del rango de las matrices N
        ug(1+N*(s-1):s*N) = mod(A1*(vg(1+N*(s-1):s*N)+A2(:,2)),256);
        ub(1+N*(s-1):s*N) = mod(A1*(vb(1+N*(s-1):s*N)+A2(:,3)),256);
    else
        y = round(mod(det(A2),256));
        if (y~=0)&&(gcd(y,2)==1) % Condición: A2 es singular
            s = s+1;             % Codificación
            ur(1+N*(s-1):s*N) = mod(A2*(vr(1+N*(s-1):s*N)+A1(:,1)),256);
            ug(1+N*(s-1):s*N) = mod(A2*(vg(1+N*(s-1):s*N)+A1(:,2)),256);
            ub(1+N*(s-1):s*N) = mod(A2*(vb(1+N*(s-1):s*N)+A1(:,3)),256);
        end
    end
end
  % Inicializamos las matrices XNR, XNG y XNB
XNR = XR;
XNG = XG;
XNB = XB;
  % Pasar ur,ug,ub (vector columna) a XNR,XNG,XNB (matriz)
for i=1:n1
    for j=1:n2
        XNR(i,j) = ur((i-1)*n2+j);
        XNG(i,j) = ug((i-1)*n2+j);
        XNB(i,j) = ub((i-1)*n2+j);
    end
end
      % Recomponemos el array de la foto, ahora codificado, a partir de las
      % matrices codificadas
XN(:,:,1) = XNR;
XN(:,:,2) = XNG;
XN(:,:,3) = XNB;
fn = uint8(XN);  % Definimos fn como el array codificado formato uint8
ff = strcat(cfoto,'.png');    % Concatenamos el nombre de la foto con .png
                              % para definir su formato
imwrite(fn,ff); %Pasa de foto en formato uint8 a formato png
T=toc;  % Terminamos cronometración
```

Una vez se ha ejecutado `codefoto1` obtenemos una foto de igual tamaño que la inicial, pero totalmente indescifrable, distribuida en sus distintas matrices de colores: `XNR,XNG,XNB`. Es importante recordar que `codefoto1` genera matrices nuevas para cada operación de producto matricial con cada subvector, en adelante analizaremos los programas con procedimientos distintos de este.

3.4 Programación de Genemat

`Genemat` es la función encargada de generar las matrices aleatorias para su uso en las multiplicaciones por los vectores de texto claro de cifrado de Hill. Sus entradas son `h,I,N,m,n,q` y `t` que son las que vienen desde el programa principal, por ejemplo `codefoto1`, ya definidas anteriormente, mientras que `t` es el valor puntual de la variable t en la función h, calculado fuera de

`genemat` (en `codefoto1`) como: `t=t0+i*dt`, donde `i` es el número de veces que se ha llamado `genemat`. En `codefoto2` y `codefoto3` como veremos posteriormente, en cambio, `i=1` y llamamos una sola vez a `genemat`, ya que no se itera la generación de matrices, sino que se calcula una sola vez y se utiliza siempre la misma.

A continuación, presentamos el código de `genemat`:

```
function [A1,A2] = genemat(h,I,N,m,n,q,t)
% h expresión de texto con las letra z y t que recoge la función iterativa
% I = [a,b]x[c,d] intervalo de definición de la h
% N orden de la matrices aleatorias (A1 y A2)
% m módulo(Z_m)
% n número de iteraciones
% q posición a partir de la cual se cuentan los decimales
% t valor actual del parámetro t
% A1 y A2 matrices pseudoaleatorias en Z_m
h = eval(strcat('@(z)',h)); %Definición de la función iterativa
  % Generamos dos matrices vacías A1 y A2
A1 = zeros(N);
A2 = A1;
  % Definimos las tolerancias tol1 y tol2 como la N-esima parte de la
  % longitud de cada tramo del mallado
tol1 = (I(2)-I(1))/N^2;
tol2 = (I(4)-I(3))/N^2;
  % Generamos un mallado de puntos rectangular contenido en el
  % rectángulo I
x = linspace(I(1)+tol1,I(2)-tol1,N);
y = linspace(I(3)+tol2,I(4)-tol2,N);
  % Generamos matrices pseudoaleatorias con valores en [a,b] y [c,d]
for r=1:N
    for s=1:N
        z = x(r)+i*y(s);
        for k=1:n
            zr = real(h(z));  % parte real de la función h(z)
            zi = imag(h(z));  % parte imaginaria de la función h(z)
            kr = floor((zr-I(1))/(I(2)-I(1)));
            zrr = zr-kr*(I(2)-I(1));      % Traemos el nuevo valor real
                                          % de la iteración al mallado
            ki = floor((zi-I(3))/(I(4)-I(3)));
            zii = zi-ki*(I(4)-I(3));      % Traemos el nuevo valor real
                                          % de la iteración al mallado
            z = zrr+i*zii;  %H(z)
        end


        A1(r,s) = zrr;    % A1(r,s) está en [a,b]
        A2(r,s) = zii;    % A2(r,s) está en [c,d]
    end
end

%Transformamos las matrices A1 y A2 en matrices pseudoaleatorias en Z_m
A1 = 10^q*A1-floor(10^q*A1);  %A1(r,s) está en [0,1]
A1 = floor(m*A1);
A2 = 10^q*A2-floor(10^q*A2);  %A2(r,s) está en [0,1]
A2 = floor(m*A2);
```

3.5 Programación de Codefoto2

Como hemos comentado anteriormente, `codefoto2` realiza el cifrado de Hill, al igual que `codefoto1`, pero con la diferencia de que no varía las matrices aleatorias para cada cálculo, sino que utiliza las dos mismas durante toda la ejecución. Eso significa que solamente llama a `genemat` una vez, es decir, fuera de ningún bucle.

```
function [XNR,XNG,XNB]=codefoto2(foto,cfoto,h,I,N,m,n,q,t0,dt)

tic
f = imread(foto);
X = double(f);
XR = X(:,:,1);
XG = X(:,:,2);
XB = X(:,:,3);

[n1,n2] = size(XR);
vr  = [];
vg  = [];
vb  = [];
for i=1:n1
    for j=1:n2
        vr = [vr;XR(i,j)];
        vg = [vg;XG(i,j)];
        vb = [vb;XB(i,j)];
    end
end
nn = n1*n2;
k = floor(nn/N);
r = mod(nn,N);
ur = vr;
ug = vg;
ub = vb;
s = 0;
t = t0;
[A1,A2] = genemat(h,I,N,m,n,q,t);
y = round(mod(det(A1),256));
while (y==0)||(gcd(y,2)==2)
    t = t+dt;
    [A1,A2] = genemat(h,I,N,m,n,q,t);
    y = round(mod(det(A1),256));
end

if r>0
    [A_1,A_2] = genemat(h,I,N,m,n,q,t);
    y = round(mod(det(A_1(1:r,1:r)),256));
    while (y==0)||(gcd(y,2)==2)
        t = t+dt;
        [A_1,A_2] = genemat(h,I,r,m,n,q,t);
        y = round(mod(det(A_1(1:r,1:r)),256));
    end
    Ar1 = A_1(1:r,1:r);
    Ar2 = A_2(1:r,1:r);
    ur(1+N*k:nn) = mod(Ar1*(vr(1+N*k:nn)+Ar2(:,1)),256);
    ug(1+N*k:nn) = mod(Ar1*(vg(1+N*k:nn)+Ar2(:,1)),256);
    ub(1+N*k:nn) = mod(Ar1*(vb(1+N*k:nn)+Ar2(:,1)),256);
end
```

```
while s<k
        s = s+1;
        ur(1+N*(s-1):s*N) = mod(A1*(vr(1+N*(s-1):s*N)+A2(:,1)),256);
        ug(1+N*(s-1):s*N) = mod(A1*(vg(1+N*(s-1):s*N)+A2(:,2)),256);
        ub(1+N*(s-1):s*N) = mod(A1*(vb(1+N*(s-1):s*N)+A2(:,3)),256);
end
XNR = XR;
XNG = XG;
XNB = XB;
for i=1:n1
    for j=1:n2
        XNR(i,j) = ur((i-1)*n2+j);
        XNG(i,j) = ug((i-1)*n2+j);
        XNB(i,j) = ub((i-1)*n2+j);
    end
end
XN(:,:,1) = XNR;
XN(:,:,2) = XNG;
XN(:,:,3) = XNB;
fn = uint8(XN);
ff = strcat(cfoto,'.png');
imwrite(fn,ff);
T = toc;
```

3.6 Programación de Codefoto3

Finalmente, el sistema de codificación que trataremos será `codefoto3`, en el que, cómo ya hemos comentado, implementa un cifrado de Hill con matrices constantes igual que `codefoto2` además de una permutación aleatoria de píxeles.

```
function [XNR,XNG,XNB] = codefoto3(foto,cfoto,h,I,N,m,n,q,t0,dt)
tic
I0 = I;
f = imread(foto);
X = double(f);
XR = X(:,:,1);
XG = X(:,:,2);
XB = X(:,:,3);

[n1,n2] = size(XR);
  % Implementamos la permutación de las matrices
I = (1:n1)-1;
J = (1:n2)-1;
ii = round(floor(n1/2));
cont = 0;
primo1 = [];
while cont<2
    if gcd(ii,n1)==1
        primo1 = [primo1;ii];
        cont = cont+1;
    end
     ii = ii-1;
end
ii = round(floor(n2/2));
cont = 0;
primo2 = [];
```

```
while cont<2
    if gcd(ii,n2)==1
        primo2 = [primo2;ii];
        cont = cont+1;
    end
     ii = ii-1;
end
I_ = mod(primo1(1)*I+primo1(2),n1)+1;
J_ = mod(primo2(2)*J+primo2(1),n2)+1;
for i=1:n1
    for j=1:n2
        XNR(i,j) = XR(I_(i),J_(j));
        XNG(i,j) = XG(I_(i),J_(j));
        XNB(i,j) = XB(I_(i),J_(j));
    end
end
XN(:,:,1) = XNR;
XN(:,:,2) = XNG;
XN(:,:,3) = XNB;
fn = uint8(XN);
XR = XNR;
XG = XNG;
XB = XNB;
vr  = [];
vg  = [];
vb  = [];
for i=1:n1
    for j=1:n2
        vr = [vr;XR(i,j)];
        vg = [vg;XG(i,j)];
        vb = [vb;XB(i,j)];
    end
end
  % Implementamos le cifrado de Hill con par de matrices pseudoaleatorias
  % único
nn = n1*n2;
k = floor(nn/N);
r = mod(nn,N);
ur = vr;
ug = vg;
ub = vb;
s = 0;
t = t0;
[A1,A2] = genemat(h,I0,N,m,n,q,t);
y = round(mod(det(A1),256));
while (y==0)||(gcd(y,2)==2)
    t = t+dt;
    [A1,A2] = genemat(h,I0,N,m,n,q,t);
    y = round(mod(det(A1),256));
end

if r>0
    [A_1,A_2] = genemat(h,I0,N,m,n,q,t);
    y = round(mod(det(A_1(1:r,1:r)),256));
    while (y==0)||(gcd(y,2)==2)
        t = t+dt;
        [A_1,A_2] = genemat(h,I0,r,m,n,q,t);
        y = round(mod(det(A_1(1:r,1:r)),256));
    end
```

```
    Ar1 = A_1(1:r,1:r);
    Ar2 = A_2(1:r,1:r);
    ur(1+N*k:nn) = mod(Ar1*(vr(1+N*k:nn)+Ar2(:,1)),256);
    ug(1+N*k:nn) = mod(Ar1*(vg(1+N*k:nn)+Ar2(:,1)),256);
    ub(1+N*k:nn) = mod(Ar1*(vb(1+N*k:nn)+Ar2(:,1)),256);
end
while s<k
        s = s+1;
        ur(1+N*(s-1):s*N) = mod(A1*(vr(1+N*(s-1):s*N)+A2(:,1)),256);
        ug(1+N*(s-1):s*N) = mod(A1*(vg(1+N*(s-1):s*N)+A2(:,2)),256);
        ub(1+N*(s-1):s*N) = mod(A1*(vb(1+N*(s-1):s*N)+A2(:,3)),256);
end
XNR = XR;
XNG = XG;
XNB = XB;
for i=1:n1
    for j=1:n2
        XNR(i,j) = ur((i-1)*n2+j);
        XNG(i,j) = ug((i-1)*n2+j);
        XNB(i,j) = ub((i-1)*n2+j);
    end
end
clear XN
XN(:,:,1) = XNR;
XN(:,:,2) = XNG;
XN(:,:,3) = XNB;
fn = uint8(XN);
ff = strcat(cfoto,'.png');
imwrite(fn,ff);
T=toc;
```

3.7 Programación de Decodefoto1

Tras haber enviado, hipotéticamente, la foto codificada junto con la clave de desencriptación, el receptor ejecuta `decodefoto1`, que toma como entrada la misma clave, y el nombre de la foto encriptada, y el futuro nombre de la versión en claro. `Decodefoto1` lleva a cabo la decodificación de cada subvector, es decir, realiza el proceso inverso al cifrado e Hill, descomponiendo el vector columna codificado en subvectores y multiplicando estos por las inversas de las matrices pseudoaleatorias y restando los vectores pseudoaleatorios.

```
function [XNR,XNG,XNB]=decodefoto1(cfoto,foto,h,I,N,m,n,q,t0,dt)
% cfoto nombre de la foto codificada
% foto nombre de la foto decodificada
% h expresión de texto con las letra z y t que recoge la función iterativa
% I = [a,b]x[c,d] intervalo de definición de la h
% N orden de la matrices aleatorias (A1 y A2)
% m módulo(Z_m)
% n número de iteraciones
% q posición a partir de la cual se cuentan los decimales
% t0 valor incial de la variable t
% dt incremento de la variable t
```

```
% XR, XG y XB son matrices en Z_256 correspondientes a los planos en escala
%de rojos, verdes y azules respectivamente de la fotografía codificada
% XNR, XNG y XNB son matrices en Z_256 correspondientes a los planos en
%escala de rojos, verdes y azules respectivamente de la fotografía
%decodificada
tic
% winopen(cfoto)
f = imread(cfoto);  % Asignamos el array tridimensional de colores de cfoto
      % a la variable f
X = double(f);
% Descomponemos las matrices de colores del array
XR = X(:,:,1);
XG = X(:,:,2);
XB = X(:,:,3);
[n1,n2] = size(XR);  % Definimos n1 y n2 cómo las dimensiones horizontal y
                     % vertical de las matrices X
vr  = [];
vg  = [];
vb  = [];
      % Pasar XR,XG,XB (matriz) a vr,vg,vb (vector columna)
for i=1:n1
    for j=1:n2
        vr = [vr;XR(i,j)];
        vg = [vg;XG(i,j)];
        vb = [vb;XB(i,j)];
    end
end
      % Definimos 'nn', 'k' y 'r'
nn = n1*n2;
k = floor(nn/N);
r = mod(nn,N);
ur = vr;
ug = vg;
ub = vb;
s = 0;
t = t0;  % Inicializamos 't'
if r>0  % En primer caso tratamos los términos restantes del vector columna
    [A_1,A_2] = genemat(h,I,N,m,n,q,t);  % Llamamos a la función genemat
    y = round(mod(det(A_1(1:r,1:r)),256));  % Calculamos el determinante
    while (y==0)||(gcd(y,2)==2)  % Condiciones de no singularidad
        t = t+dt;
        [A_1,A_2] = genemat(h,I,r,m,n,q,t);  % Recalculamos matrices
        y = round(mod(det(A_1(1:r,1:r)),256)); % R ecalculamos el
                                               % determinante
    end
    Ar1 = A_1(1:r,1:r);   % Calculamos las matrices reducidas para los
                          % términos restantes
    Ar2 = A_2(1:r,1:r);
    Br1 = invmod(Ar1,256);% Calculamos la inversa de la matriz reducida Ar1
    ur(1+N*k:nn) = mod(Br1*(vr(1+N*k:nn))-Ar2(:,1),256);
    ug(1+N*k:nn) = mod(Br1*(vg(1+N*k:nn))-Ar2(:,1),256);
    ub(1+N*k:nn) = mod(Br1*(vb(1+N*k:nn))-Ar2(:,1),256);
end
while s<k
    t = t+dt;
    [A1,A2] = genemat(h,I,N,m,n,q,t);  % Generamos matrices
    y = round(mod(det(A1),256));  % Calculamos el determinante
    if (y~=0)&&(gcd(y,256)==1)    % Condición: A1 es singular
        B1 = invmod(A1,256);      % Calculamos la inversa de A1
```

```
        s = s+1;                    % Decodificación
        ur(1+N*(s-1):s*N) = mod(mod(B1*(vr(1+N*(s-1):s*N)),256)-
A2(:,1),256);
        ug(1+N*(s-1):s*N) = mod(mod(B1*(vg(1+N*(s-1):s*N)),256)-
A2(:,2),256);
        ub(1+N*(s-1):s*N) = mod(mod(B1*(vb(1+N*(s-1):s*N)),256)-
A2(:,3),256);
    else    % En caso de que A1 no tiene inversa, calculamos la de A2 ya
            % que esta será la matriz multiplicadora
        y = round(mod(det(A2),256));
        if (y~=0)&&(gcd(y,256)==1)% A2 es singular
            B2 = invmod(A2,256);  % Calculamos la inversa de A2
            s = s+1;              % Decodificación
            ur(1+N*(s-1):s*N) = mod(mod(B2*(vr(1+N*(s-1):s*N)),256)-
A1(:,1),256);
            ug(1+N*(s-1):s*N) = mod(mod(B2*(vg(1+N*(s-1):s*N)),256)-
A1(:,2),256);
            ub(1+N*(s-1):s*N) = mod(mod(B2*(vb(1+N*(s-1):s*N)),256)-
A1(:,3),256);
        end
    end
end

XNR = XR;
XNG = XG;
XNB = XB;
  % Reconstruimos las matrices de la foto descifrada
for i=1:n1
    for j=1:n2
        XNR(i,j) = ur((i-1)*n2+j);
        XNG(i,j) = ug((i-1)*n2+j);
        XNB(i,j) = ub((i-1)*n2+j);
    end
end
  % Recuperamos el array de la foto
XN(:,:,1) = XNR;
XN(:,:,2) = XNG;
XN(:,:,3) = XNB;
fn = uint8(XN);
ff = strcat(foto,'.png');
imwrite(fn,ff);
toc
```

3.8 Programación de Decodefoto2

`Decodefoto2` decodifica la fotografía generada tras la ejecución de `codefoto2`. Al igual que con `codefoto1` y `decodefoto1`, el receptor ejecuta `decodefoto2` que realiza el proceso inverso a `codefoto2`.

```
function [XNR,XNG,XNB]=decodefoto2(cfoto,foto,h,I,N,m,n,q,t0,dt)
tic

f = imread(cfoto);

X = double(f);
XR = X(:,:,1);
```

```
XG = X(:,:,2);
XB = X(:,:,3);

[n1,n2] = size(XR);
vr  = [];
vg  = [];
vb  = [];
for i=1:n1
    for j=1:n2
        vr = [vr;XR(i,j)];
        vg = [vg;XG(i,j)];
        vb = [vb;XB(i,j)];
    end
end
nn = n1*n2;
k = floor(nn/N);
r = mod(nn,N);
ur = vr;
ug = vg;
ub = vb;
s = 0;
t = t0;
[A1,A2] = genemat(h,I,N,m,n,q,t);
y = round(mod(det(A1),256));
while (y==0)||(gcd(y,2)==2)
    t = t+dt;
    [A1,A2] = genemat(h,I,N,m,n,q,t);
    y = round(mod(det(A1),256));
end
B1 = invmod(A1,256);

if r>0
    [A_1,A_2] = genemat(h,I,N,m,n,q,t);
    y = round(mod(det(A_1(1:r,1:r)),256));
    while (y==0)||(gcd(y,2)==2)
        t = t+dt;
        [A_1,A_2] = genemat(h,I,r,m,n,q,t);
        y = round(mod(det(A_1(1:r,1:r)),256));
    end
    Ar1 = A_1(1:r,1:r);
    Ar2 = A_2(1:r,1:r);
    Br1 = invmod(Ar1,256);
    ur(1+N*k:nn) = mod(Br1*(vr(1+N*k:nn))-Ar2(:,1),256);
    ug(1+N*k:nn) = mod(Br1*(vg(1+N*k:nn))-Ar2(:,1),256);
    ub(1+N*k:nn) = mod(Br1*(vb(1+N*k:nn))-Ar2(:,1),256);
end
while s<k
    s = s+1;
    ur(1+N*(s-1):s*N) = mod(B1*(vr(1+N*(s-1):s*N))-A2(:,1),256);
    ug(1+N*(s-1):s*N) = mod(B1*(vg(1+N*(s-1):s*N))-A2(:,2),256);
    ub(1+N*(s-1):s*N) = mod(B1*(vb(1+N*(s-1):s*N))-A2(:,3),256);
end
XNR = XR;
XNG = XG;
XNB = XB;
for i=1:n1
    for j=1:n2
        XNR(i,j) = ur((i-1)*n2+j);
        XNG(i,j) = ug((i-1)*n2+j);
```

```
        XNB(i,j) = ub((i-1)*n2+j);
    end
end
XN(:,:,1) = XNR;
XN(:,:,2) = XNG;
XN(:,:,3) = XNB;
fn = uint8(XN);

nombre = strcat(foto,'.png');
imwrite(fn,nombre);
T = toc;
```

3.9 Programación de Decodefoto3

Análogamente a los casos anteriores, el programa de decodificación del tercer método `decodefoto3` es básicamente el programa inverso al de codificación. Hay que recordar que esta vez se realiza la operación inversa al cifrado de Hill primero, y a continuación la inversa a la permutación.

```
function [XNR,XNG,XNB] = decodefoto3(cfoto,foto,h,I,N,m,n,q,t0,dt)
tic
I0 = I;

f = imread(cfoto);

X = double(f);
XR = X(:,:,1);
XG = X(:,:,2);
XB = X(:,:,3);

[n1,n2] = size(XR);
vr  = [];
vg  = [];
vb  = [];
for i=1:n1
    for j=1:n2
        vr = [vr;XR(i,j)];
        vg = [vg;XG(i,j)];
        vb = [vb;XB(i,j)];
    end
end
nn = n1*n2;
k = floor(nn/N);
r = mod(nn,N)
ur = vr;
ug = vg;
ub = vb;
s = 0;
t = t0;
[A1,A2] = genemat(h,I0,N,m,n,q,t);
y = round(mod(det(A1),256));
while (y==0)||(gcd(y,2)==2)
    t = t+dt;
    [A1,A2] = genemat(h,I0,N,m,n,q,t);
    y = round(mod(det(A1),256));
end
```

```
B1 = invmod(A1,256);
if r>0
    [A_1,A_2] = genemat(h,I0,N,m,n,q,t);
    y = round(mod(det(A_1(1:r,1:r)),256));
    while (y==0)||(gcd(y,2)==2)
        t = t+dt;
        [A_1,A_2] = genemat(h,I0,r,m,n,q,t);
        y = round(mod(det(A_1(1:r,1:r)),256));
    end
    Ar1 = A_1(1:r,1:r);
    Ar2 = A_2(1:r,1:r);
    Br1 = invmod(Ar1,256)
    ur(1+N*k:nn) = mod(Br1*(vr(1+N*k:nn))-Ar2(:,1),256);
    ug(1+N*k:nn) = mod(Br1*(vg(1+N*k:nn))-Ar2(:,1),256);
    ub(1+N*k:nn) = mod(Br1*(vb(1+N*k:nn))-Ar2(:,1),256);
end
while s<k
    s = s+1;
    ur(1+N*(s-1):s*N) = mod(B1*(vr(1+N*(s-1):s*N))-A2(:,1),256);
    ug(1+N*(s-1):s*N) = mod(B1*(vg(1+N*(s-1):s*N))-A2(:,2),256);
    ub(1+N*(s-1):s*N) = mod(B1*(vb(1+N*(s-1):s*N))-A2(:,3),256);
end
XNR = XR;
XNG = XG;
XNB = XB;
for i=1:n1
    for j=1:n2
        XNR(i,j) = ur((i-1)*n2+j);
        XNG(i,j) = ug((i-1)*n2+j);
        XNB(i,j) = ub((i-1)*n2+j);
    end
end
XN(:,:,1) = XNR;
XN(:,:,2) = XNG;
XN(:,:,3) = XNB;
XR = XNR;
XG = XNG;
XB = XNB;
I = (1:n1)-1;
J = (1:n2)-1;
ii = round(floor(n1/2));
cont = 0;
primo1 = [];
while cont<2
    if gcd(ii,n1)==1
        primo1 = [primo1;ii];
        cont = cont+1;
    end
     ii = ii-1;
end
ii = round(floor(n2/2));
cont = 0;
primo2 = [];
while cont<2
    if gcd(ii,n2)==1
        primo2 = [primo2;ii];
        cont = cont+1;
    end
     ii = ii-1;
```

```
end
I_ = mod(primo1(1)*I+primo1(2),n1)+1;
J_ = mod(primo2(2)*J+primo2(1),n2)+1;
for i=1:n1
    for j=1:n2
        XNR(I_(i),J_(j)) = XR(i,j);
        XNG(I_(i),J_(j)) = XG(i,j);
        XNB(I_(i),J_(j)) = XB(i,j);
    end
end
XNN(:,:,1) = XNR;
XNN(:,:,2) = XNG;
XNN(:,:,3) = XNB;
fn = uint8(XNN);

ff = strcat(foto,'.png');
imwrite(fn,ff);

T = toc;
```

3.10 Programas del algoritmo de Diffie-Hellman

A continuación, expondremos los programas de Diffie-Hellman, en los cuales implementamos el propio algoritmo a Matlab entre dos usuarios, que son evidentemente los dos participantes del intercambio de información, siendo el usuario 1 presumiblemente el emisor, y el 2 el receptor. Los programas están desarrollados en el orden que a priori se deberían ejecutar, de acuerdo con el algoritmo en sí.

El algoritmo está diseñado para codificar por clave pública una de las variables que se emplean en el cifrado de Hill, en principio `t0` o `dt`, ya que ambas son números fraccionarios. Aunque también se pueden emplear para cifrar el resto de las variables ya que, aunque estas sean números enteros y nuestro algoritmo de Diffie-Hellman está diseñado para dar como salida una clave compartida fraccionaria (que llamamos `t`) que proviene de una norma matricial, los participantes pueden establecer por convenio usar solo la parte entera de esta, o redondear, o usar cierta posición decimal.

3.10.1 Programación de Usuario1a_DH

En primer lugar el usuario 1 ejecuta `Usuario1a_DH(p,g,a)` para obtener su clave pública `x`, que es un vector, que a continuación manda al usuario 2.

```
function x=Usuario1a_DH(p,g,a)
x=mod(g.^a,p);    % Calculamos el vector x que es el que le enviamos al
                  % usuario 2
end
```

3.10.2 Programación de Usuario2a_DH

Tras recibir `x`, el usuario 2 ejecuta `Usuario2a_DH` obteniendo la clave secreta compartida `t`.

```
function [t,y]=Usuario2a_DH(p,g,x,b)
for i=1:numel(x)  % Tras recibir xx del usuario 1, generamos la matriz y
```

```
                % donde una columna se compone de cada término de x
                % elevado al término de a respectivo b esa posición
    for j=1:numel(b)
        y(i,j) = mod(x(i)^b(j),p);
    end
end
t=norm(y);  % clave secreta Usuario 2
end
```

3.10.3 Programación de Usuario2b_DH

A continuación, ejecuta `Usuario2b_DH` para calcular su clave pública `xx`, formato vector, y enviársela al usuario 1.

```
function xx=Usuario2b_DH(p,g,b)
xx = mod(g.^b,p); % Calculamos el vector xx que es el que le enviamos al
                  % usuario 1
end
```

3.10.4 Programación de Usuario1b_DH

De forma simétrica, el usuario 1 recibe `xx` y calcula la clave secreta compartida `t`.

```
function [t,yy]=Usuario1b_DH(p,xx,a)
for i=1:numel(a)  % Tras recibir xx del usuario 2, generamos la matriz yy
                  % donde una columna se compone de cada término de xx
                  % elevado al término de a respectivo a esa posición
     for j=1:numel(xx)
        yy(i,j) = mod(xx(j)^a(i),p);
     end
end
t=norm(yy);  % clave secreta Usuario 1
end
```

4. Análisis y valoración de resultados

Tras implementar los programas, es el momento de analizar su rendimiento y efectividad para codificar imágenes, en definitiva, comprobar su fiabilidad a través de la robustez que presenta a un ataque externo y la aleatoriedad lograda por el algoritmo a partir de dos parámetros fundamentales: la uniformidad y la independencia de resultados. En primer lugar, ejecutaremos una serie de pruebas de codificación con distintas imágenes, visualizando los resultados de los tres métodos, y sacando las conclusiones respectivas a la viabilidad de cada uno, evaluando la robustez del cifrado contra la variabilidad de la clave. A continuación, analizaremos las matrices pseudoaleatorias generadas, evaluando las distribuciones de valores generadas, por medio de una serie de pruebas estadísticas de uniformidad y aleatoriedad. De la misma forma, evaluaremos la aleatoriedad de los programas que consigan resultados aparentemente aleatorios, es decir, donde la foto codificada sea prácticamente intrazable a la original a simple vista, sometiéndolas a las pruebas de aleatoriedad y uniformidad. Finalmente, llevaremos a cabo un ensayo de los tiempos de ejecución de cada programa con respecto al tamaño de los documentos que se codifican y con ellos evaluaremos cuales son más preferibles a utilizar según las condiciones en las que codificaremos.

4.1 Ensayos de aplicación de los algoritmos

Vamos a ejecutar pruebas de los 3 programas sobre dos imágenes, la fotografía de Lenna, con la clave `'z^2+0.27i+t',[0,1,0,1],7,31,12,8,0,0.01` y una de Tintín, el popular personaje de cómic, con la clave `'z^2+0.5*z-0.3+2i+t',[0,1,0,1],8,37,6,10,1,0.2` obteniendo los siguientes resultados:

Figura 12. Imagen de Lenna original

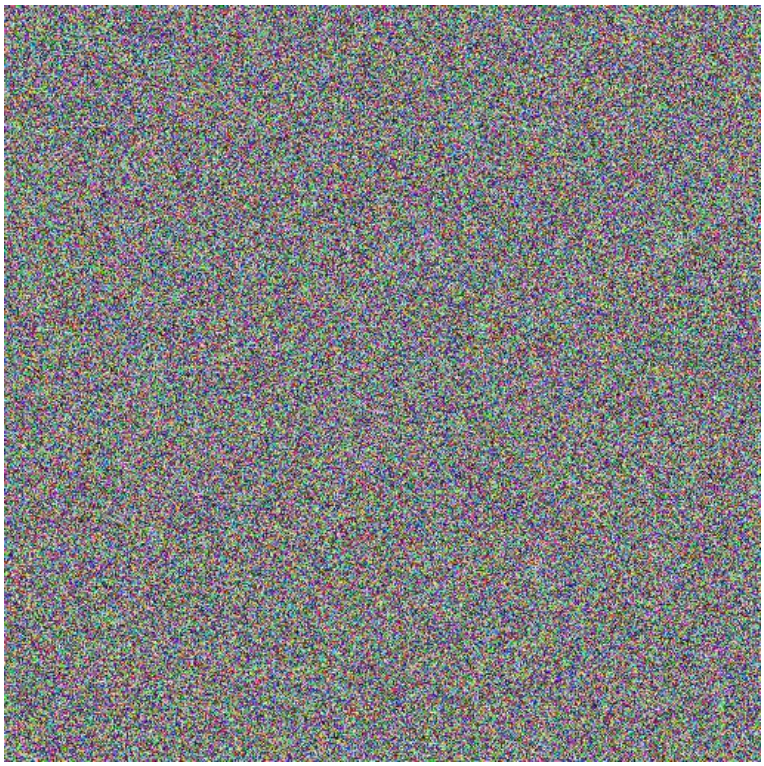

Figura 13. Imagen de Lenna codificada por codefoto1

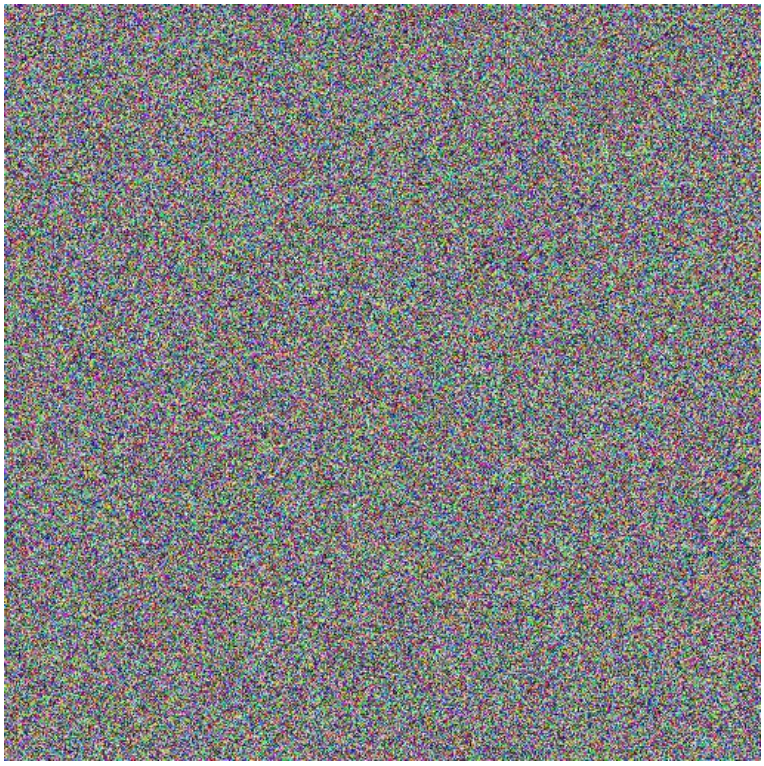

Figura 14. Imagen de Lenna codificada por codefoto2

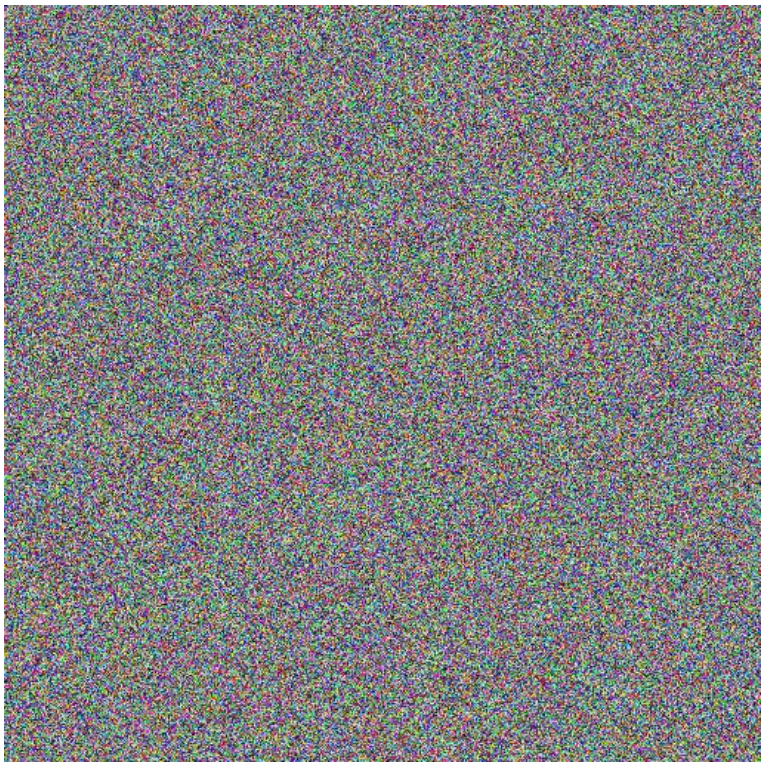

Figura 15. Imagen de Lenna codificada por codefoto3

Figura 16. Imagen de Tintin original

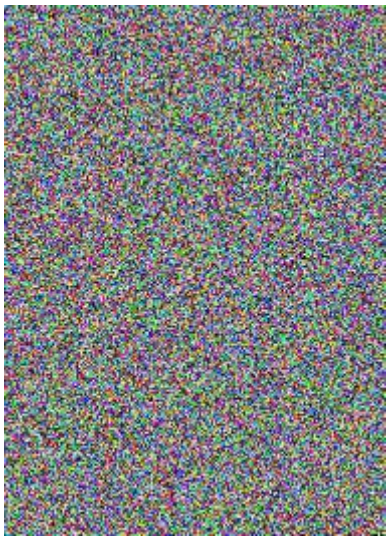

Figura 17. Imagen de Tintin codificada por codefoto1

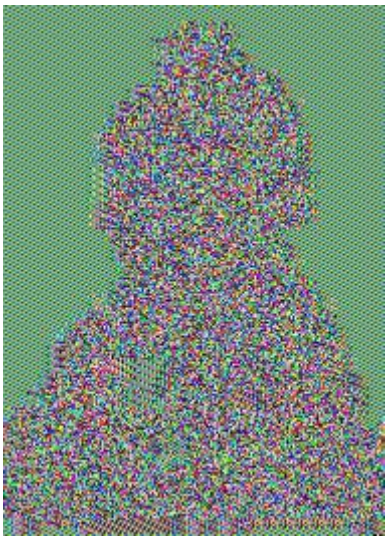

Figura 18. Imagen de Tintin codificada por codefoto2

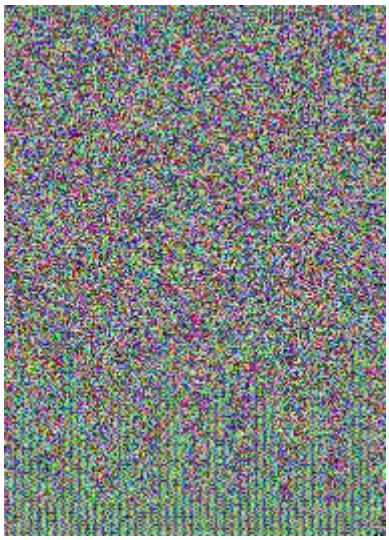

Figura 19. Imagen de Tintin codificada por codefoto3

Podemos observar que todos los algoritmos funcionan a la hora de codificar fotos visualmente irregulares, cómo la de Lenna. Pero el algoritmo de `codefoto2` no sirven con imágenes que presentan patrones de colores regulares, cómo la de Tintín, en la que observamos claramente la silueta del personaje contra, donde se situaba el fondo blanco en la foto original, por lo tanto, no ofrece aleatoriedad de resultados siempre y serán utilizados sólo en codificaciones con ciertas condiciones. Esto es porque, al ser la matriz y el vector pseudoaleatorios siempre iguales, cuando codifican una sección regular, es decir, del mismo color o en patrones definidos de varios colores, el cifrado de Hill resulta en un producto de los vectores a codificar, que al ser el vector columna de esa sección constante o periódico, son constantes, por la matriz pseudoaleatoria que es siempre la misma y luego una suma del vector que es constante también. Por tanto, da como resultado una sección periódica en la misma sección de la foto regular. Con `codefoto3` se subsana mayoritariamente este problema, pero se siguen observando regiones regulares en la foto, en este caso en la parte inferior de la imagen. En definitiva, `codefoto1` es el algoritmo de referencia en la codificación ya que es el más eficaz y que menos problemas presenta.

Para más demostración de ello, vamos a ejecutar los programas sobre una imagen de un fondo negro y una de un código de barras, con claves que son, respectivamente, `'sin(3*z)-0.7i+t',[-1,1,-1,1],8,17,7,7,0,0.1` y `'exp(pi*z)+3*z-0.01i+0.5*t',[0,2,-1,0],9,23,7,6,2,0.1`:

Figura 20. Imagen de fondo negro original

Figura 21. Imagen de fondo negro codificada por codefoto1

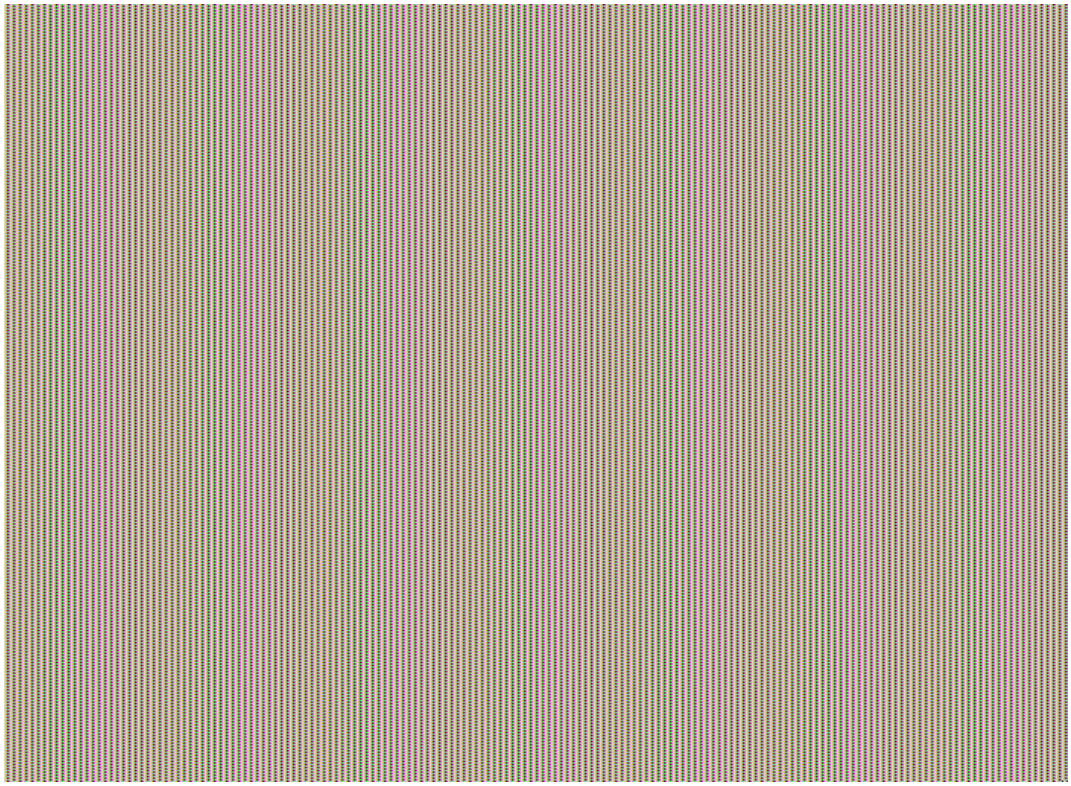

Figura 22. Imagen de fondo negro codificada por codefoto2

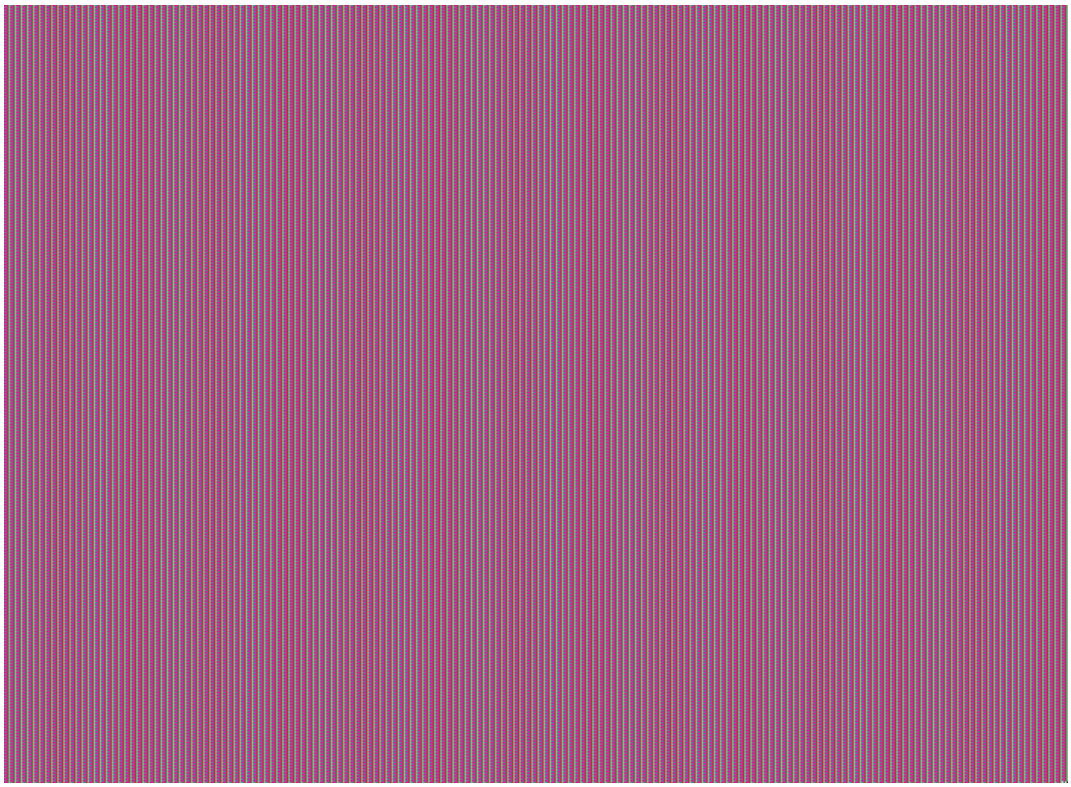

Figura 23. Imagen de fondo negro codificada por codefoto3

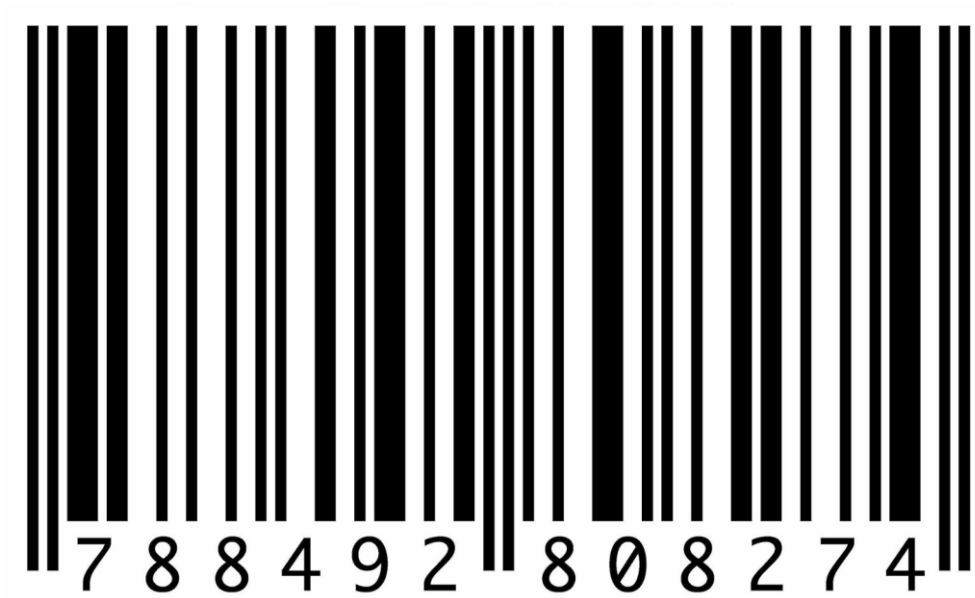

Figura 24. Imagen de código de barras original

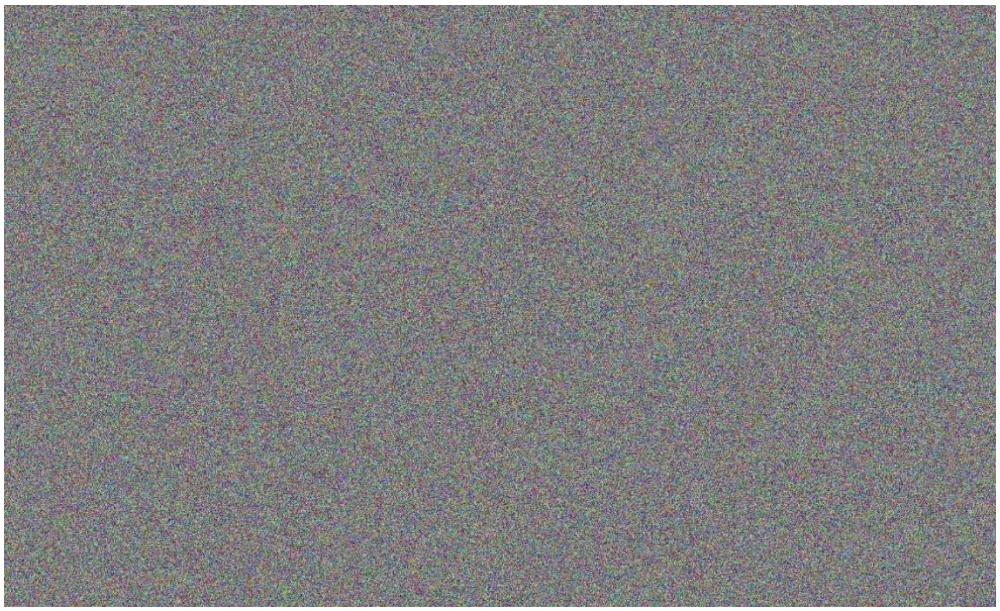

Figura 25. Imagen de código de barras codificada por codefoto1

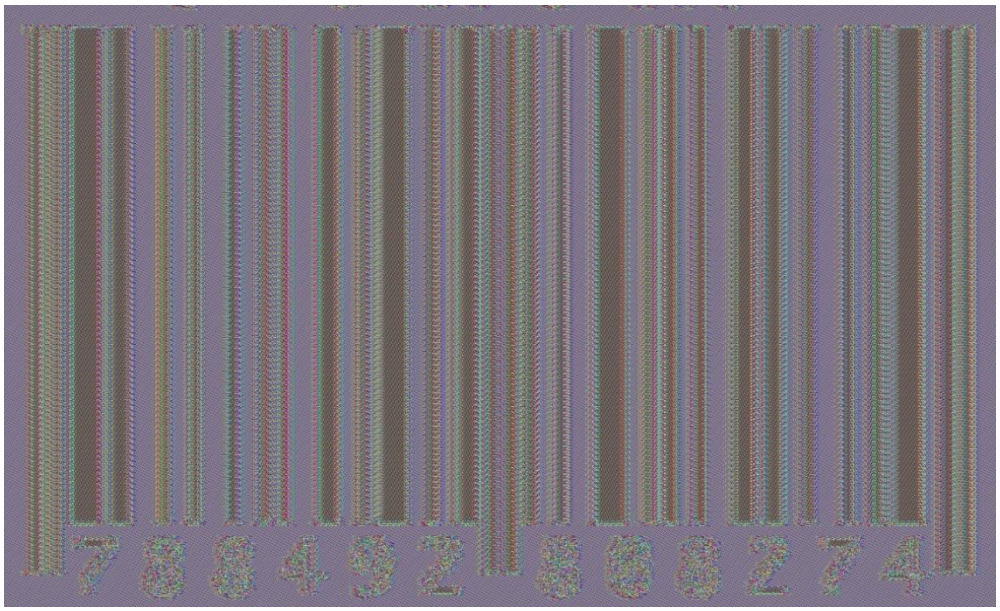

Figura 26. Imagen de código de barras codificada por codefoto2

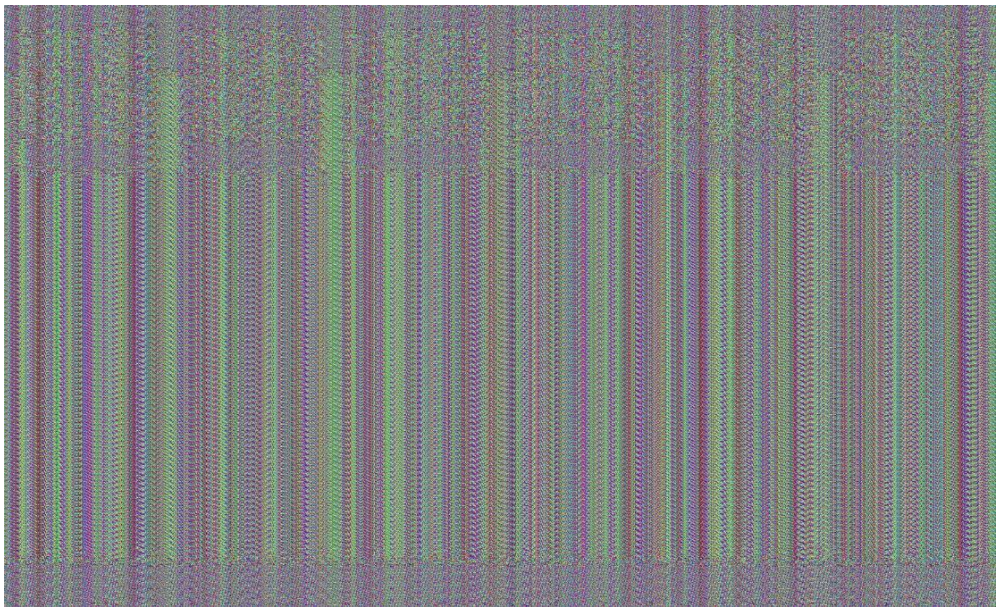

Figura 27. Imagen de código de barras codificada por codefoto3

Definitivamente, `codefoto2` y, en menor medida, `codefoto3` son poco eficaces para codificar imágenes con tramos de colores regulares, como se ve en las codificaciones de la imagen de negro, así como en la del código de barras, con `codefoto2` la codificación todavía muestra el contorno de la imagen original, mientras que codofoto3 es un poco más útil ya que no es inmediata averiguar la foto original, pero sigue mostrando regularidad en la codificación.

A continuación, observamos histogramas de las distribuciones de color de la foto de Lenna original y codificada por `codefoto1`. Se observa claramente la uniformidad en la distribución de la imagen codificada, a diferencia de la original, donde la gama de colores es más restringida, como es natural.

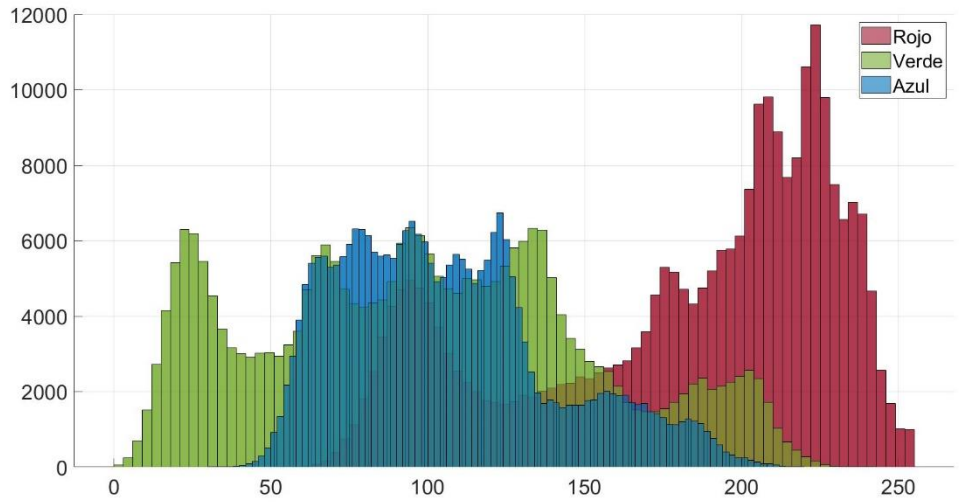

Figura 28. Histograma de la distribución de intensidades de la imagen de Lenna original

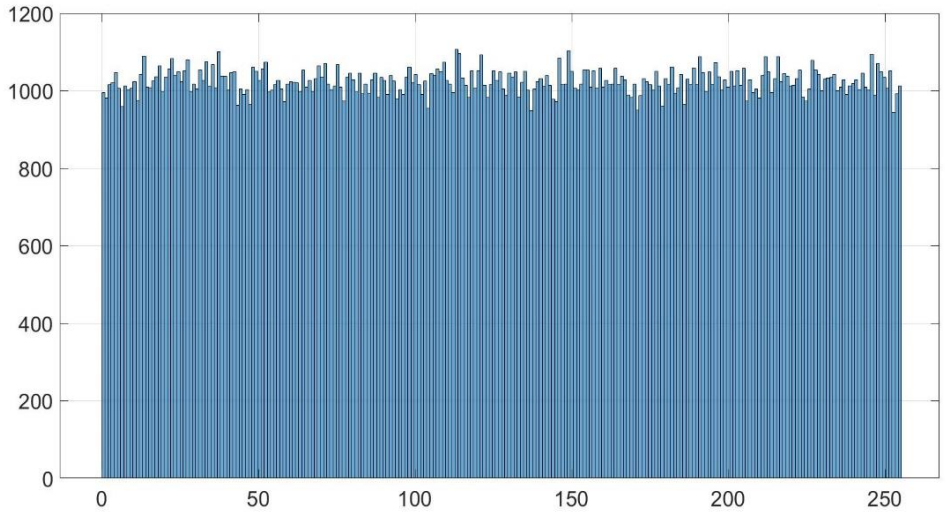

Figura 29. Histograma de la distribución de intensidades de azules de la imagen de Lenna codificada

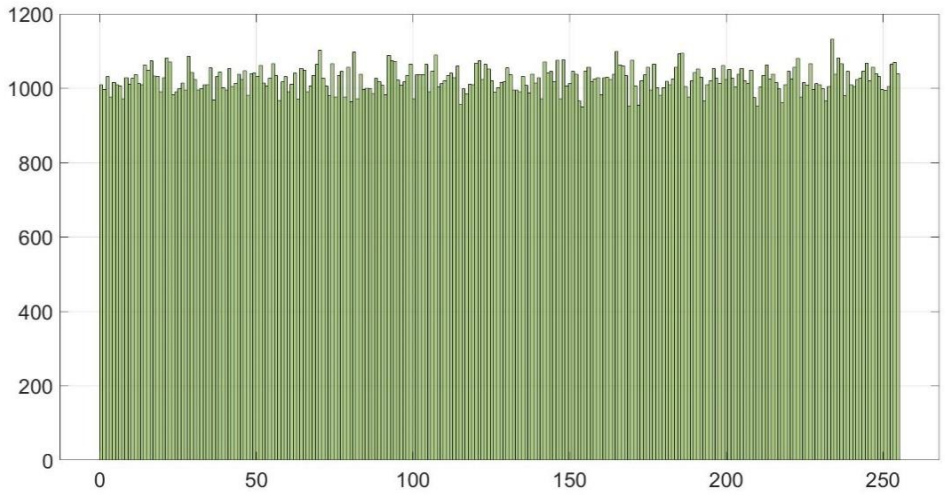

Figura 30. Histograma de la distribución de intensidades de verdes de la imagen de Lenna codificada

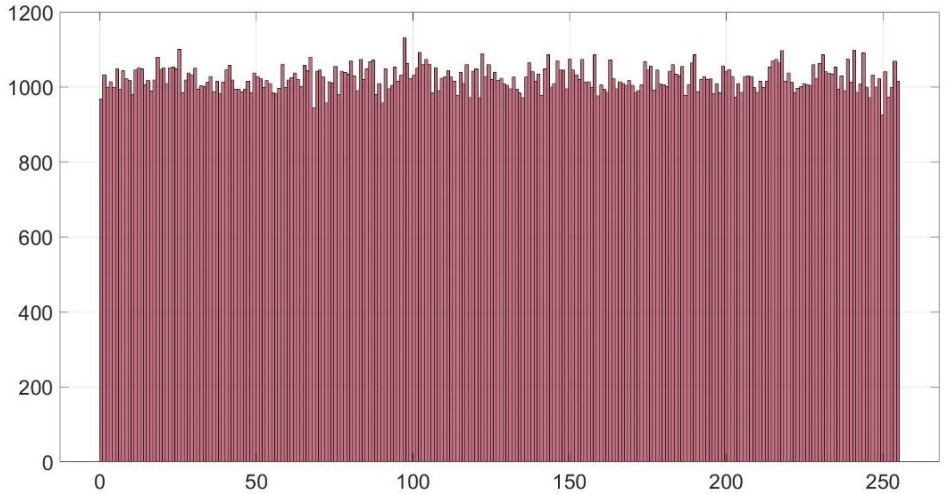

Figura 31. Histograma de la distribución de intensidades de rojos de la imagen de Lenna codificada

4.1.1 Robustez y fiabilidad del algoritmo

Los algoritmos presentan un alto grado de variabilidad ante cambios de las condiciones iniciales, es decir, su valor de condición es muy alto, luego es lo contrario de un problema bien condicionado, lo cual es muy deseable para un sistema de codificación.

Con las variables de tipo finito, como `I`, `N`, `m`, `q`, un cambio de una cifra en su valor hace que el descifrado sea completamente imposible, pero más allá de esto, con las variables continuas de la clave, como son t0, dt, con una variación del orden de 10^{-14} en el valor de la clave ya no es posible la decodificación, se muestra a continuación:

Hemos codificado la foto de lena con `codefoto1('lenna.jpg','lenna_crf', 'tan(pi*z)-2*z-0.2i+t',[-1,2,-1,2],6,23,5,10,0,0.15)`, se ha decodificado con decodefoto1, primero con la clave correcta, y luego con la misma clave pero cambiando la variable `dt` de 0.15 a 0.150000000000001 (una diferencia de 1×10^{-14}), es decir, la codificamos con `codefoto1('lenna.jpg','lenna_crf','tan(pi*z)-2*z-0.2i+t',[-1,2,-1,2],6,23, 5,10,0,0.150000000000001)`:

Figura 32. Imagen de Lenna decodificada con decodefoto1 con dt = 0.15

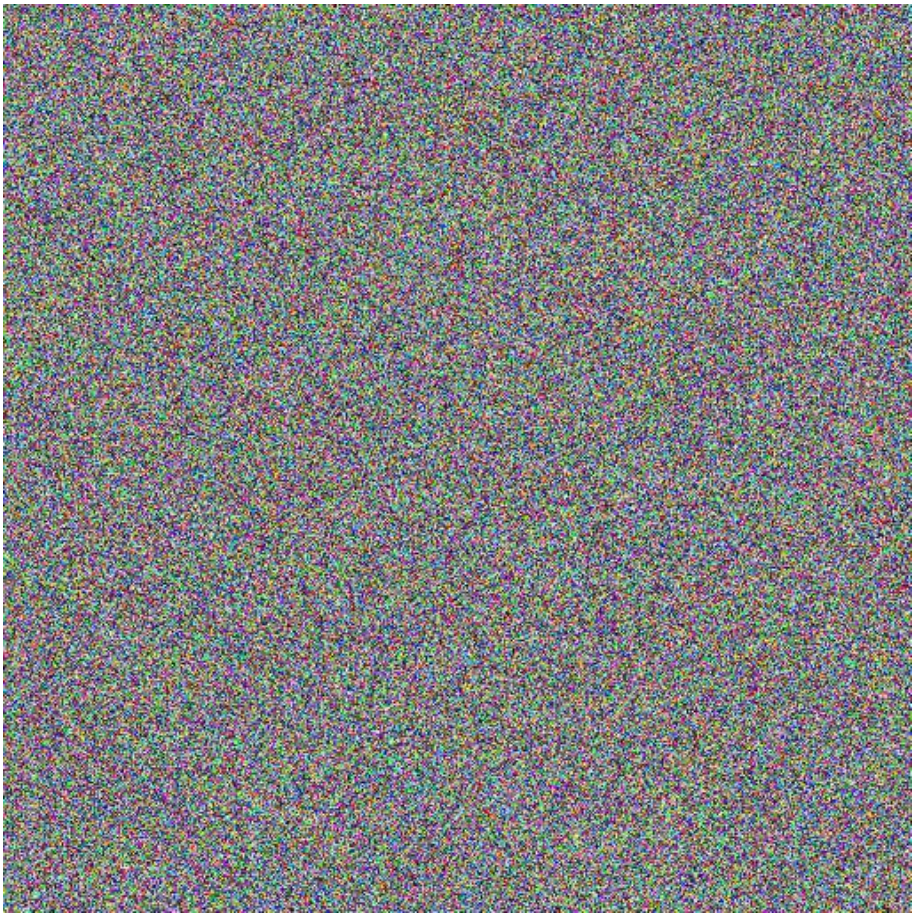

Figura 33. Imagen de Lenna decodificada con decodefoto1 con dt = 0.150000000000001

Es por tanto un sistema muy robusto, con una clave es muy difícil de estimar por prueba y error, ya que no es discernible que se produzca aproximación con valores muy cercanos a la clave correcta, ni los más cercanos posibles.

4.2 Análisis estadístico de las matrices pseudoaleatorias

Para comprobar con rigor la aleatoriedad del algoritmo generador debemos evaluar la uniformidad, el grado en que los datos siguen una generación uniforme (con los valores estandarizados U[0,1]) y la independencia, el grado en el que un dato observado no depende del dato anterior, es decir no se aprecia un patrón en los resultados. Utilizaremos la función: `u = matvec(XN,a,b,c,d)`, que transforma el intervalos `[a,b,c,d]` de la matriz `XN` en un vector columna con los términos normalizados (programación en el Anexo B). Aplicaremos la prueba de Chi-cuadrado, la de Kolmogorov-Smirnov, la de correlación serial, la de prueba serial y diversas pruebas de rachas. Escribiremos estos tests en Matlab (mostrados en el Anexo A) y los implementaremos para evaluar matrices pseudoaleatorias generadas por `genemat`. Se las aplicaremos a una matriz pseudoaleatoria generada por `genemat('z^2+0.27i+t',[0,1,0,1],7,31,12,8,0)`. A continuación, tenemos la matriz pseudoaleatoria generada, y las distribuciones absoluta, relativa y acumulada de esta. [9, 11]

```
[A1,A2] = genemat('z^2+0.27i+t',[0,1,0,1],7,31,12,8,0)
```

A1 =

22	1	29	2	24	27	25
19	6	17	7	30	14	4
8	20	10	15	5	10	5
17	0	23	1	21	8	27
27	2	28	18	11	16	30
25	12	23	13	15	24	14
15	20	19	15	19	16	14

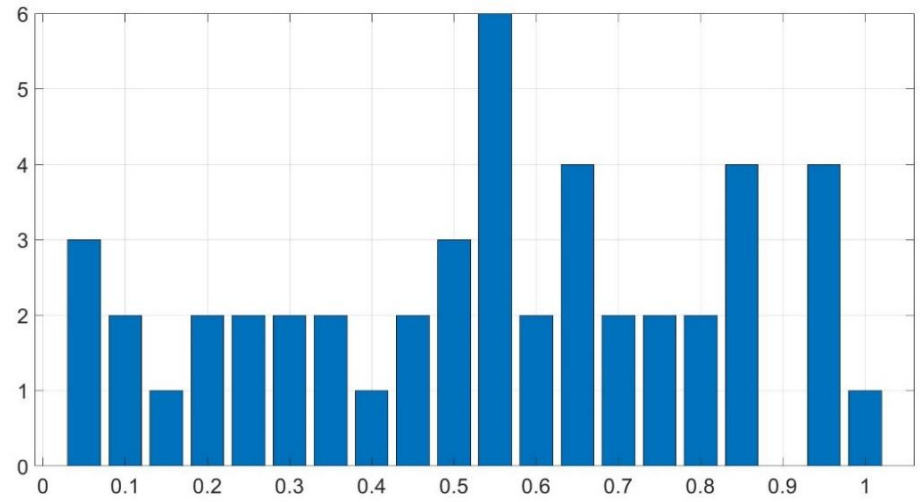

Figura 34. Histograma de frecuencias absolutas de la matriz pseudoaleatoria

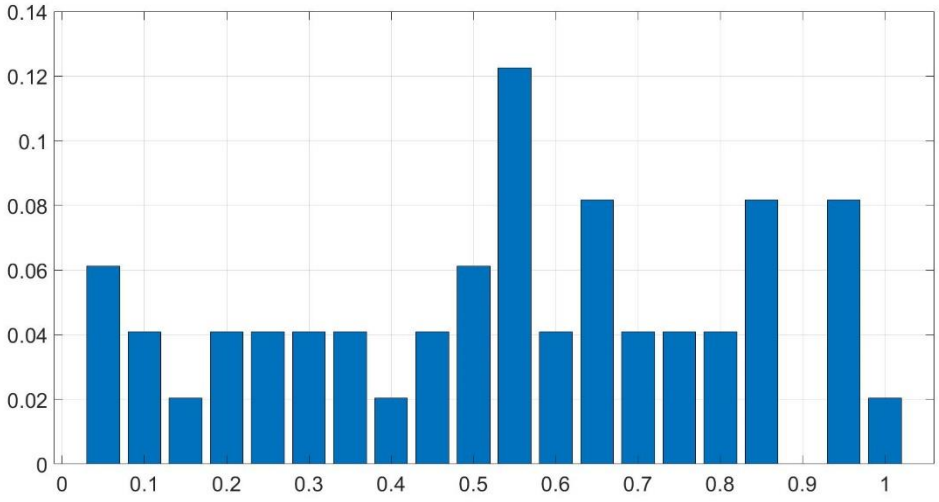

Figura 35. Histograma de frecuencias relativas de la matriz pseudoaleatoria

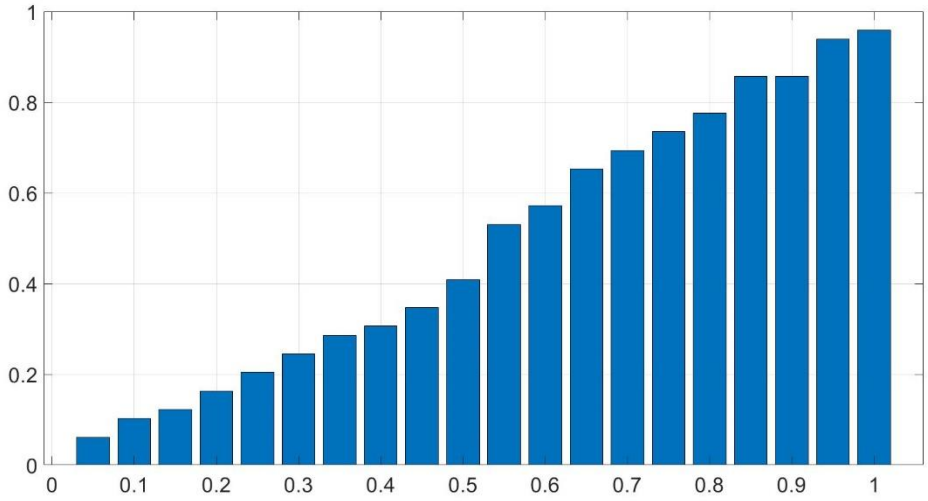

Figura 36. Histograma de frecuencias relativas acumuladas de la matriz pseudoaleatoria

4.2.1 Prueba de Chi-cuadrado sobre la matriz

En un histograma con una distribución de *k* intervalos, y siendo O_i la frecuencia observada y E_i la frecuencia observada para la *i*-esima celda, comparamos O_i y E_i por la siguiente expresión:

$$T = \chi_0^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \qquad [4.1]$$

Si el tamaño de la muestra es n entonces $E_i = \frac{n}{k}$. Es conocido que para n grande T sigue una distribución χ^2 con $k-1$ grados de libertad. Para un nivel de significación α se acepta que los datos siguen una distribución χ^2 con $k-1$ grados de libertad si $T < \chi^2_{[\alpha,k-1]}$.

Aplicado a la matriz pseudoaleatoria:

Calculamos Chi-cuadrado con *k*=10 intervalos. χ_0^2=3.6122 es menor que $\chi^2_{[0.1;9]}$=14.6837, por lo tanto, no hay evidencia de que los números no son uniformes a un nivel de significación de α=0.1.

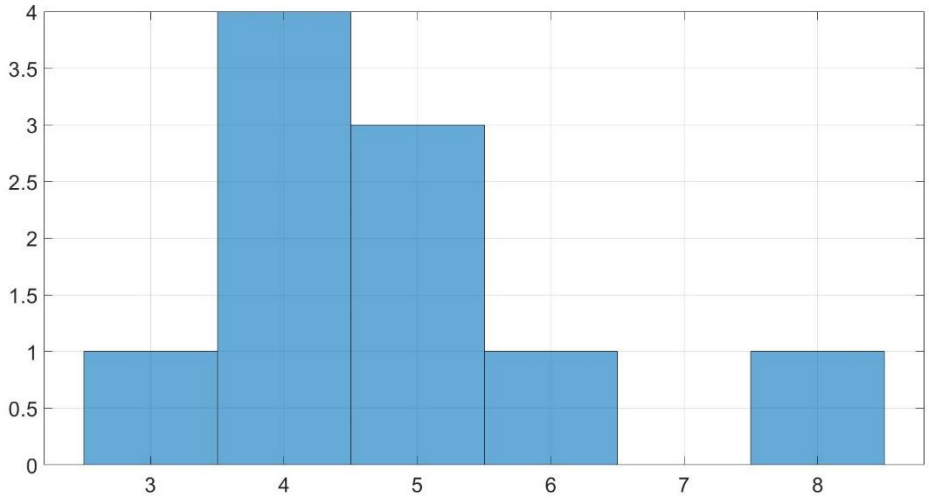

Figura 37. Frecuencias observadas de los datos de la matriz pseudoaleatoria

4.2.2 Prueba de Kolmogorov-Smirnov sobre la matriz

La prueba K-S nos permite decidir si una muestra de *n* observaciones es de una distribución continua particular. Para el caso de una distribución uniforme en el intervalo [0,1] y una muestra $\{u_1, u_2, \dots, u_n\}$ tal que $u_{j-1} < u_j$ se definen

$$D^+ = \max_j \left\{\frac{j}{n} - u_j\right\} \qquad D^- = \max_j \left\{u_j - \frac{j-1}{n}\right\} \qquad [4.2]$$

Para un nivel de significación de α, si los valores de D^+ y D^- son menores que $D_{[\alpha;n]}$ (valor de la tabla de Kolmogorov-Smirnov) entonces se dice que las observaciones provienen de la distribución respectiva con un nivel de significación determinado α.

Aplicado a la matriz pseudoaleatoria:

$D^+ = 0.0354$

$D^- = 0.1197$

Ambos menores que $D_{[0.1;49]} = 0.17128$, por lo tanto, decimos que no hay evidencia de que los números no sean uniformes a un nivel de significación de α=0.1.

4.2.3 Prueba de correlación serial sobre la matriz

Cuando la covarianza entre los valores de una muestra es no nula, entonces las dos variables que definen la distribución son dependientes. Sin embargo, la situación inversa no es cierta, es decir, que la covarianza sea cero no implica que sean independientes. En una secuencia de números aleatorios podemos calcular la autocovarianza con desplazamiento *k*, que es la covarianza entre números *k*-distantes, es decir, entre x_i y x_i+*k*. La denotamos por R_k y viene definida por:

$$R_k = \frac{1}{n-k}\sum_{i=1}^{n-k}\left(x_i - \frac{1}{2}\right)\left(x_{i+k} - \frac{1}{2}\right) \quad [4.3]$$

donde x_i es el i-esimo número aleatorio uniforme de la secuencia.

Con una n grande, R_k se distribuye como una normal con media 0 y varianza 1/[144(n-k)]. El intervalo de confianza para la autocovarianza al $100(1-\alpha)\%$ es

$$R_k \pm {}^{z_{1-\frac{\alpha}{2}}}\Big/_{\left(12\sqrt{n-k}\right)} \quad [4.4]$$

Decimos que la secuencia tiene correlación significativa cuando este intervalo no incluye el cero, aplicable solamente para $k \geq 1$.

Aplicado a la matriz pseudoaleatoria (donde R_ es R_k y IR_ el intervalo de confianza):

R_ =	IR_ =	
-0.0218	-0.0454	0.0017
0.0184	-0.0054	0.0423
-0.0024	-0.0264	0.0217
0.0109	-0.0134	0.0353
0.0061	-0.0185	0.0307
0.0072	-0.0177	0.0321
0.0163	-0.0089	0.0415
-0.0203	-0.0458	0.0052
0.0067	-0.0192	0.0325

Todos los intervalos incluyen el cero y podemos asumir todas las covarianzas como estadísticamente no significativas al nivel de confianza α=0.05 o $Z_{0.975}$=1.958. Es decir, no rechazamos la hipótesis de que no haya correlación significativa.

4.2.4 Prueba serial sobre la matriz

Para demostrar la uniformidad en dos dimensiones o más, en este caso dos, aplicamos la prueba serial, en la que discretizamos el espacio entre 0 y 1 en K^2 celdas de igual área. Si la muestra es de tamaño n, el número de pares no solapados (x_1, x_2), (x_3, x_4), ..., que se definen son n/2. El número de puntos que caen en cada celda, en el caso ideal, se espera que sean $\frac{n/2}{K^2}$. Aplicamos Chi-cuadrado para encontrar la desviación entre lo observado y lo esperado, siendo los grados de libertad $K^2 - 1$. Se puede extender la prueba a k-dimensiones.

Aplicado a matriz pseudoaleatoria:

Prueba serial k=5

O = 0 1 2 1 0
0 0 2 2 2
1 0 1 1 0
1 0 3 0 1
2 0 1 1 1

T = 18.6250

T (chi-cuadrado) es menor que $\chi^2_{[0.1;24]} = 33.2$, por lo tanto, aceptamos que los números son uniformes en dos dimensiones a un nivel de significación de $\alpha = 0.1$.

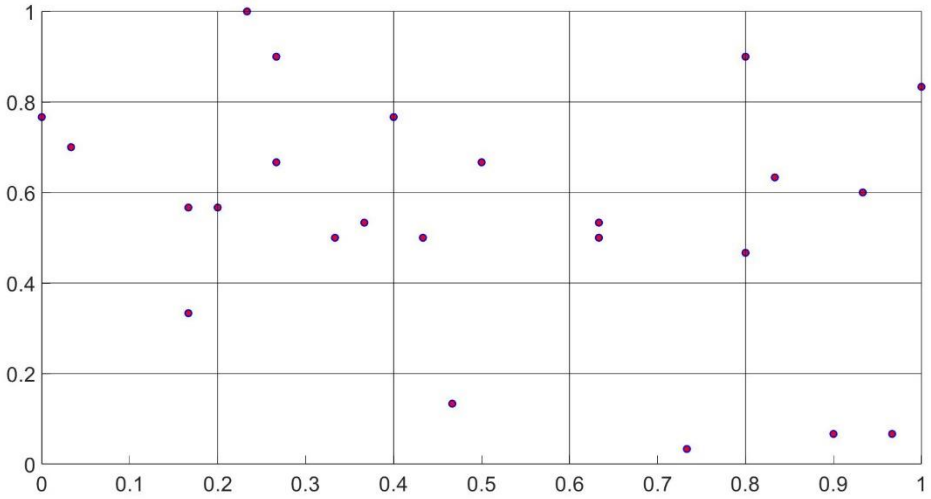

Figura 38. Distribución de pares no solapados de la matriz pseudoaleatoria

4.2.5 Pruebas de rachas sobre la matriz

Cuando tenemos conjuntos de números en secuencia ordenadas, definitivamente no son aleatorios, pero podrían pasar las pruebas de uniformidad de Chi-cuadrado o Kolmogorov-Smirnov. En este caso aplicamos las pruebas de rachas que sí toman en consideración la forma en que distribuye el conjunto de números.

4.2.5.1 Número de rachas crecientes y decrecientes (matriz)

Llamamos rachas crecientes en las que el número siguiente es mayor que el anterior, y decrecientes en las que es menor.

Aplicado a la matriz pseudoaleatoria:

Número de rachas crecientes/decrecientes

R = 35
Z0 = 0.9207
mu = 32.3333

Como Z_0=0.9207 es menor que $Z_{\alpha/2}$=1.645, no rechazamos la hipótesis de independencia a un nivel de significación de α=0.1.

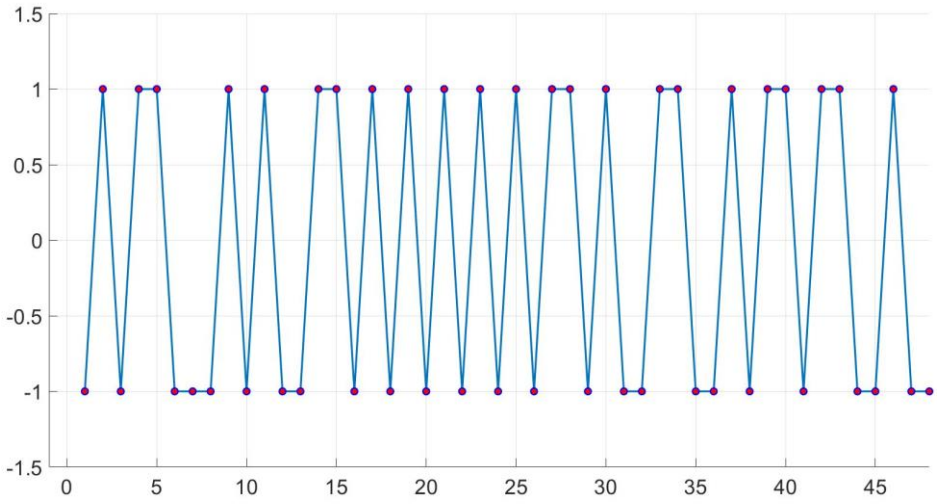

Figura 39. Distribución de las rachas crecientes y decrecientes en la muestra dada por la matriz pseudoaleatoria

4.2.5.2 Número de rachas por encima y por debajo de la media (matriz)

Para una secuencia de números normalizada, definimos rachas por encima y debajo de la media a aquellas de números mayores y menores que 0.5, la media de la uniforme, respectivamente.

Aplicado a la matriz pseudoaleatoria:

Número de rachas por encima/por debajo de la media

R = 32
mu = 24.9898
sigma = 3.4619
n1 = 25
n2 = 24

Como $Z_0 = 2.0250$ es menor que $Z_{\alpha/2} = 2.24$, por lo tanto, no rechazamos la hipótesis de independencia a un nivel de significación de $\alpha = 0.025$.

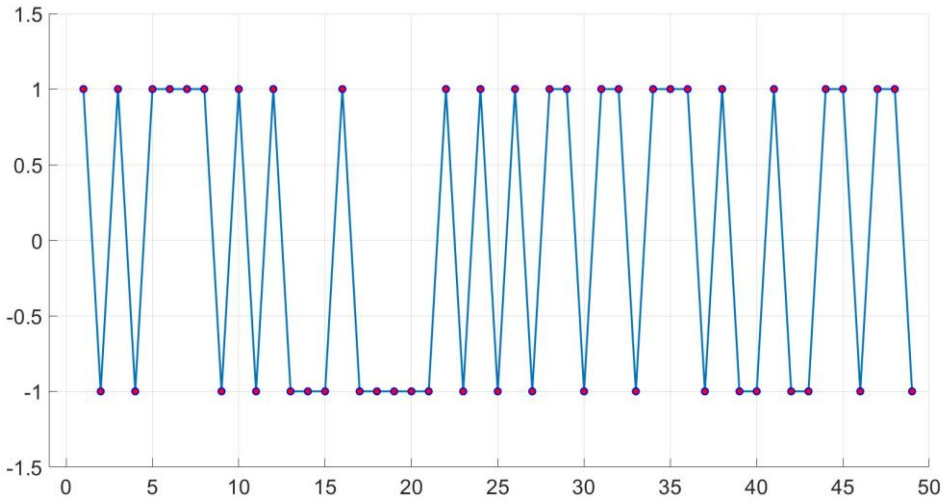

Figura 40. Distribución de las rachas por encima y por debajo de la media en la muestra dada por la matriz pseudoaleatoria

4.2.5.3 Longitud de rachas crecientes y decrecientes (matriz)

En las anteriores pruebas de rachas hemos cuantificado el volumen de variación de las rachas, pero también es importante su longitud. Para una secuencia independiente de rachas crecientes/decrecientes los valores esperados de *Yi* están dados por:

$$E(Y_i) = \frac{2}{(i+3)!}[N(i^2+3i+1) - (i^3+3i^2-i-4)] \text{ si } i \leq N-2 \qquad [4.5]$$

$$E(Y_i) = \frac{2}{N!} \text{ si } i = N-1 \qquad [4.6]$$

Aplicado a la matriz pseudoaleatoria (O son los valores observados y E los observados):

Longitud de rachas crecientes/decrecientes

O = 23
12
E = 20.5000
11.8333

Como $\chi_0^2 = 0.3072$ es menor que $\chi^2_{[0.1;1]} = 2.71$, no rechazamos la hipótesis de independencia a un nivel de significación de α=0.1.

4.2.5.4 Longitud de rachas sobre y bajo la media (matriz)

Para las rachas sobre y bajo la media tenemos:

$$E(Y_i) = \frac{Nw_i}{E(I)} \quad [4.7]$$

$$w_i = \left(\frac{n_1}{N}\right)^i \left(\frac{n_2}{N}\right) + \left(\frac{n_1}{N}\right)\left(\frac{n_2}{N}\right)^i \quad [4.8]$$

$$E(I) = \frac{n_1}{n_2} + \frac{n_2}{n_1} \quad [4.9]$$

$$E(A) = \frac{N}{E(I)} \quad [4.10]$$

Donde w_i es la probabilidad aproximada de que una racha tenga longitud i, $E(I)$ es la longitud promedio esperada de las rachas y $E(A)$ es el número esperado aproximado de rachas (de cualquier longitud) en la secuencia.

$$\chi_0^2 = \sum_{i=1}^{L} \frac{\left(O_i - E(Y_i)\right)^2}{E(Y_i)} \quad [4.11]$$

donde $L = N$-1 para rachas creciente/decrecientes y $L = N$ para rachas sobre/bajo la media. Comparamos y se acepta que los datos son independientes sí, con un nivel de significancia de α, $\chi_0^2 < \chi_{[\alpha,L-1]}^2$.

Aplicado a la matriz pseudoaleatoria:

Longitud de rachas por encima/por debajo de la media

O = 22	E = 12.2347
6	6.1174
4	6.1275

Como $\chi_0^2 = 8.5353$ es menor que $\chi_{[0.01;2]}^2 = 9.2104$, no rechazamos la hipótesis de independencia a un nivel de significación de α=0.1.

4.3 Análisis estadístico sobre secciones de la imagen codificada

Hemos demostrado la uniformidad y aleatoriedad de las matrices pseudoaleatorias generadas, pero, además, aplicaremos las pruebas estadísticas sobre la imagen codificada, para comprobar si, al igual que las matrices, el resultado de la operación de cifrado es estadísticamente aceptable, como debería ser el caso. Vamos a aplicar las pruebas estadísticas a una sección de la foto de Lenna codificada por `codefoto1('lenna.jpg','lenna_c','z^2+0.27i+t',[0,1,0,1],7,31,12,8,0,0.01)`, en particular, y a modo de ejemplo, a las matrices de rojos, azules y verdes correspondientes a secciones de píxeles de la imagen. En la función `matvec` introduciremos como `XN` las matrices de rojo, azules y verdes codificadas obtenidas de `codefoto1 (XNR,XNB,XNG)`. En los tests de Chi-cuadrado y Kolmogorov-Smirnov utilizaremos el intervalo [`251,257,251,257`], para el resto de las pruebas el intervalo [`250,300,250,300`]. [11]

A continuación, tenemos la distribución relativa y acumulada de la sección de la foto para cada color, con los ejes de abscisas normalizados, siendo el valor de 1 equivalente a una tonalidad de los píxeles de 255.

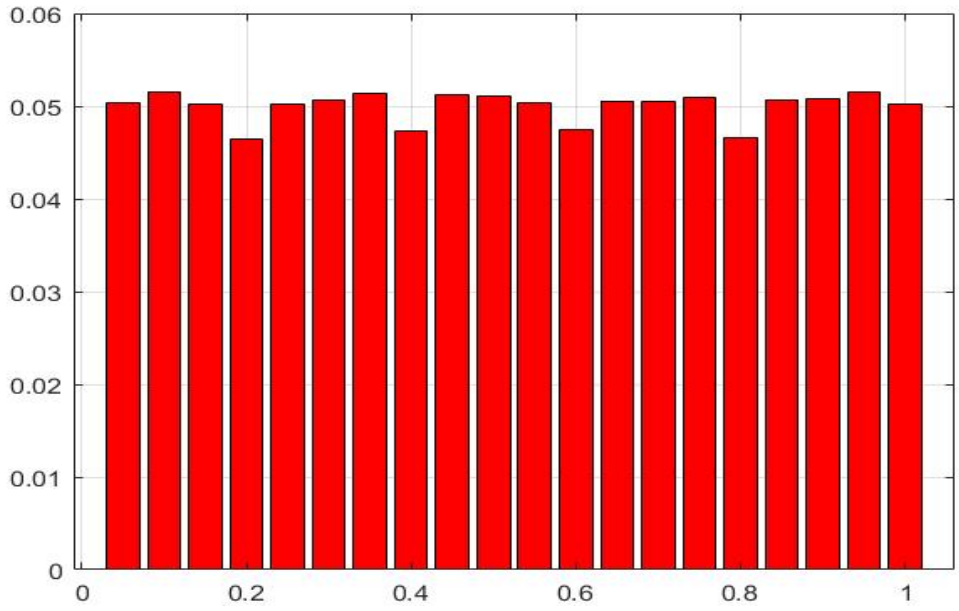

Figura 41. Histograma de frecuencias relativas de rojos

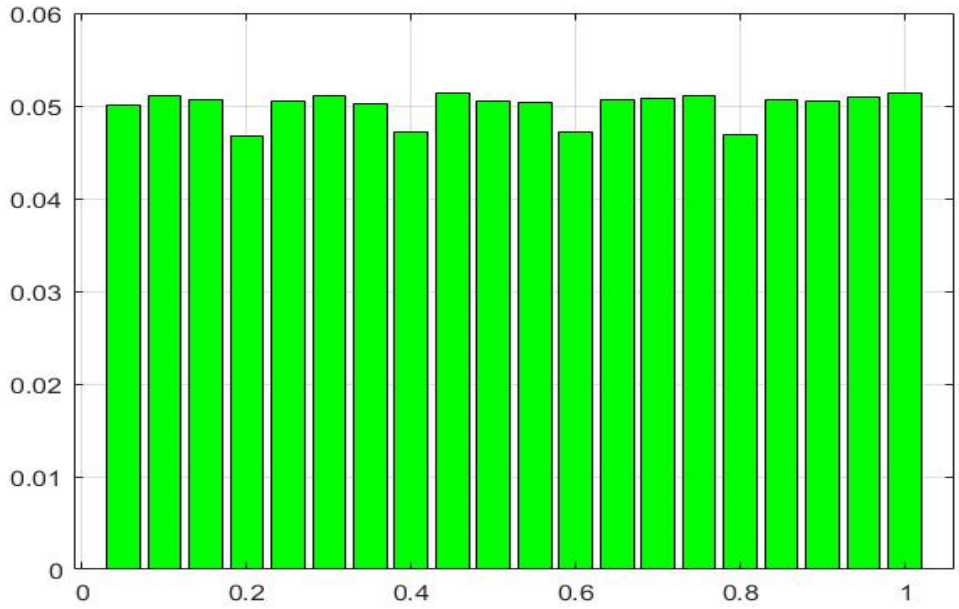

Figura 42. Histograma de frecuencias relativas de verdes

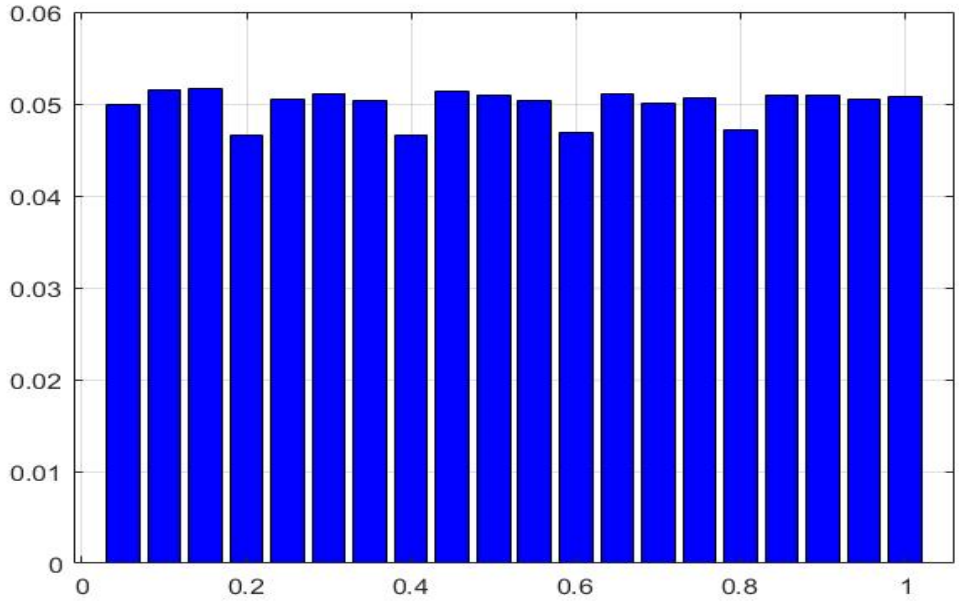

Figura 43. Histograma de frecuencias relativas de azules

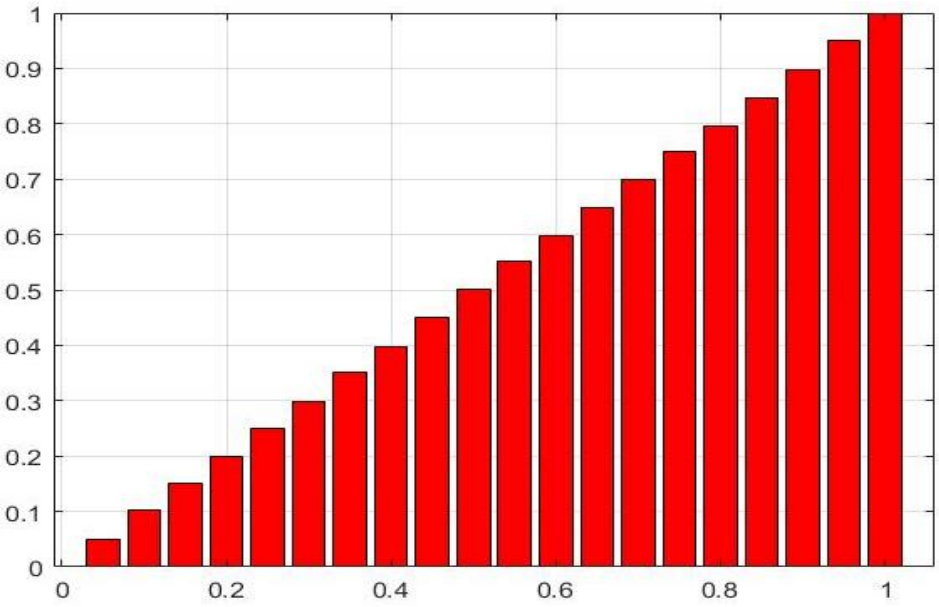

Figura 44. Gráfico de frecuencias acumuladas de rojos

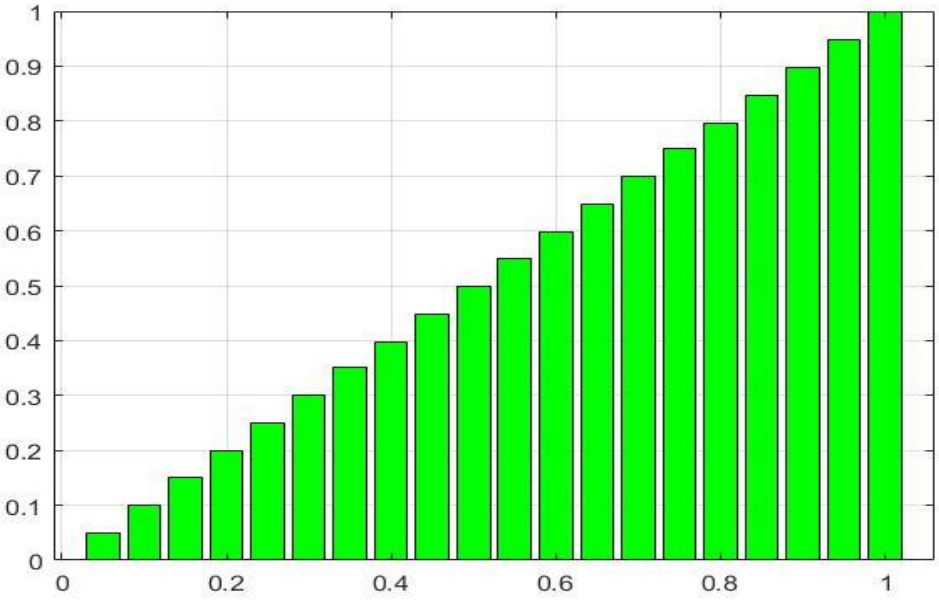

Figura 45. Gráfico de frecuencias acumuladas de verdes

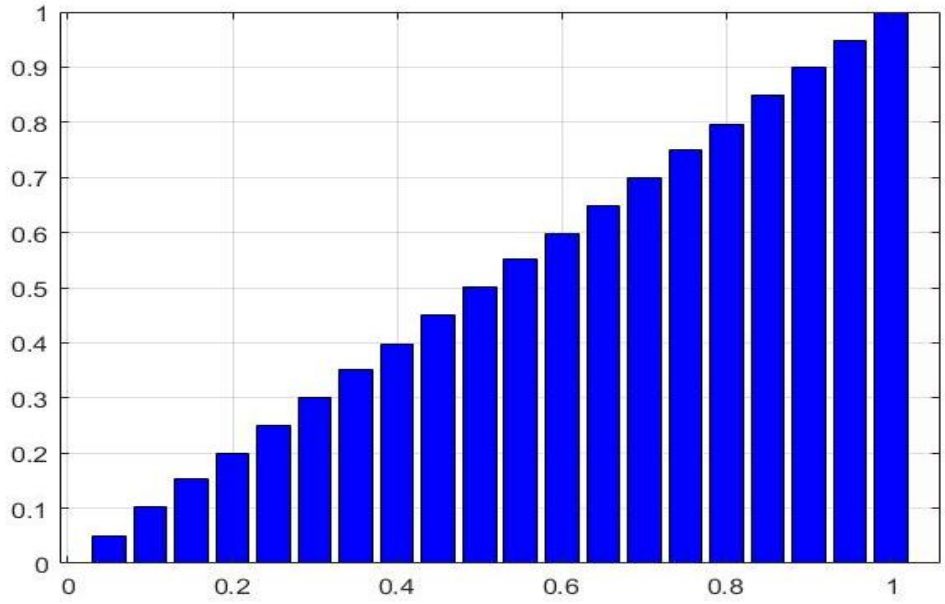

Figura 46. Gráfico de frecuencias acumuladas de azules

4.3.1 Prueba de Chi-cuadrado sobre la imagen

Aplicado a la sección de la imagen de lenna.jpg correspondiente a la sección de píxeles de la imagen definida por el intervalo [251,257,251,257] trabajando con $k = 10$ intervalos se obtienen los valores:

R matriz de intensidades de rojos

T $= 14.8776$ menor que $\chi^2_{[0.05;9]} = 16.9190$ por lo tanto no hay evidencia de que los números no son uniformes a un nivel de significación de $\alpha = 0.05$.

G matriz de intensidades de verdes

T $= 3.4490$ menor que $\chi^2_{[0.1;9]} = 14.6837$, por lo tanto, no hay evidencia de que los números no son uniformes a un nivel de significación de $\alpha = 0.1$.

B matriz de intensidades de azules

T $= 7.1224$ menor que $\chi^2_{[0.1;9]} = 14.6837$, por lo tanto, no hay evidencia de que los números no son uniformes a un nivel de significación de $\alpha = 0.1$.

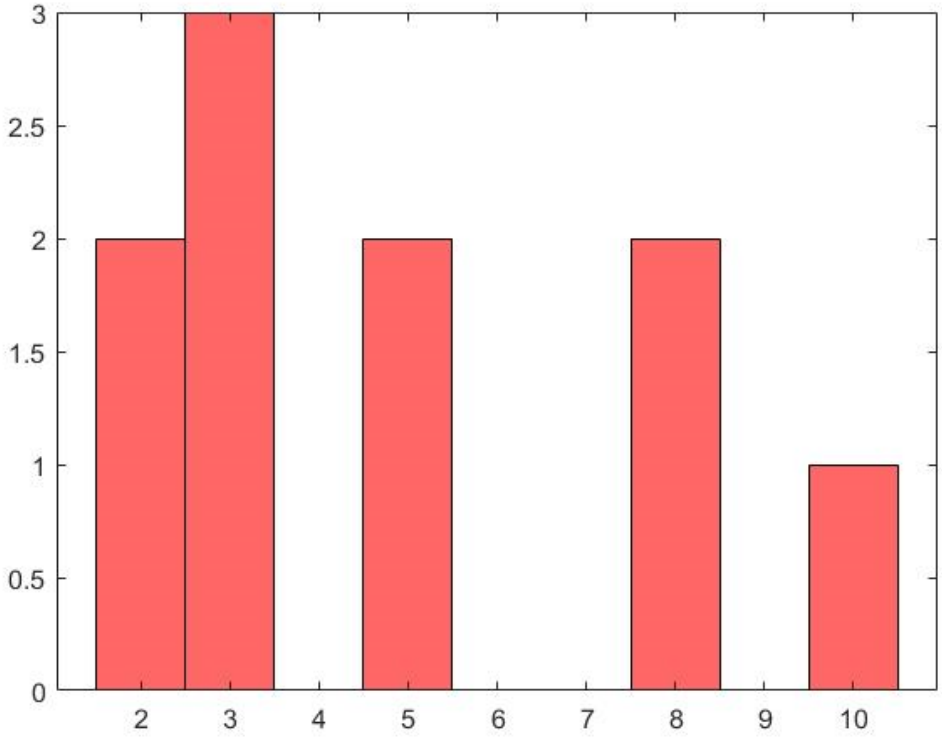

Figura 47. Frecuencias observadas de rojos

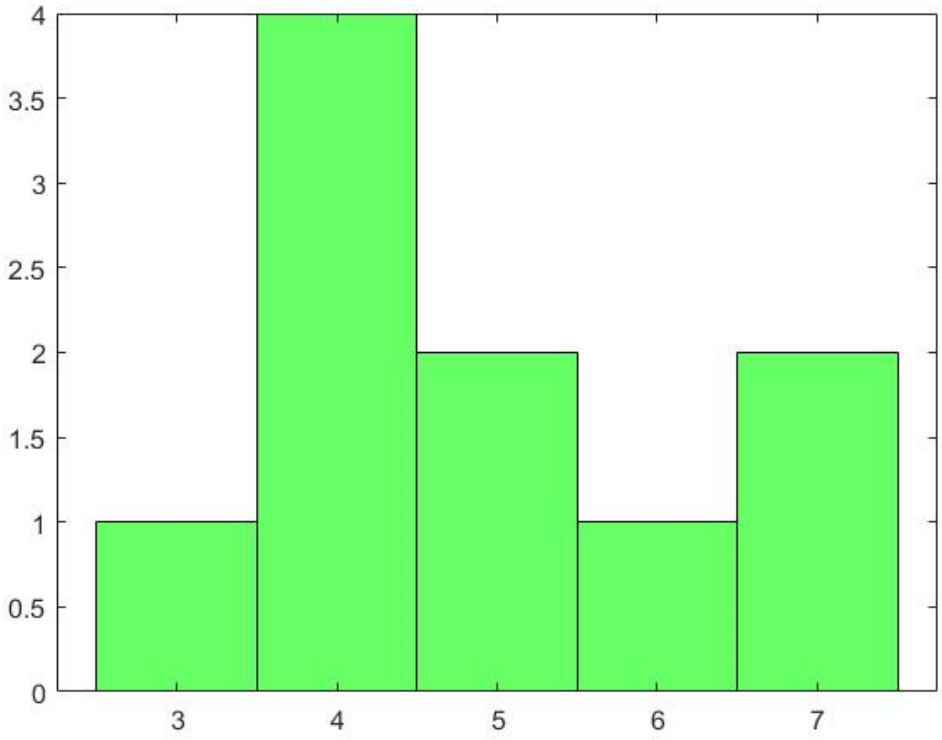

Figura 48. Frecuencias observadas de verdes

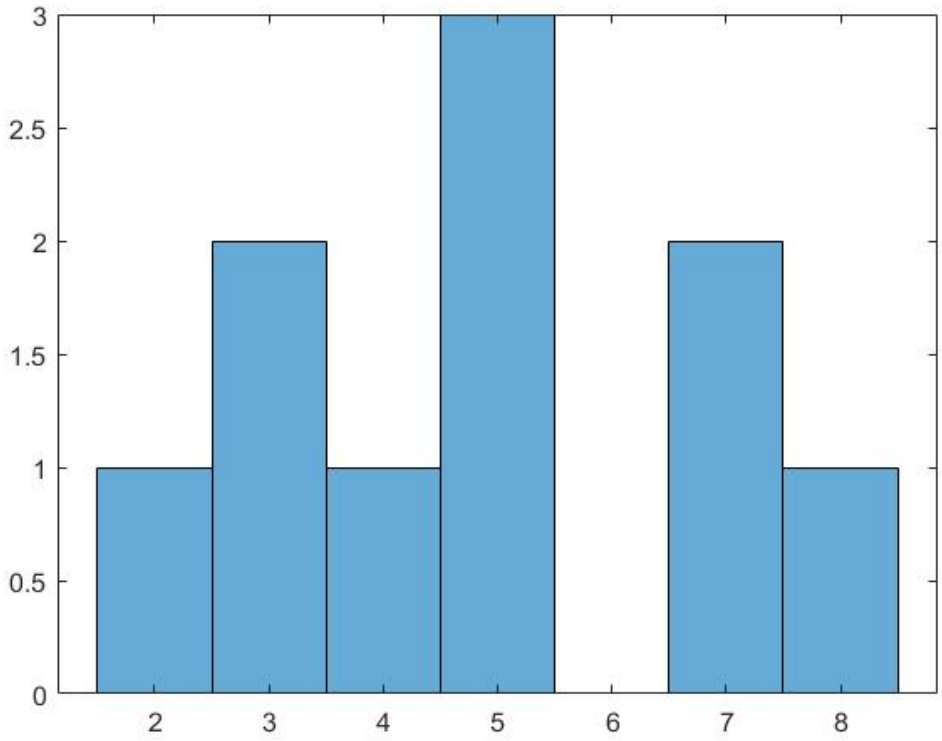

Figura 49. Frecuencias observadas de azules

4.3.2 Prueba de Kolmogorov-Smirnov sobre la imagen

Aplicado a la sección de píxeles de la imagen definida por el intervalo [251,257,251,257] se obtienen los valores:

R matriz de intensidades de rojos $D^+ = 0.0814$ y $D^- = 0.1056$

R matriz de intensidades de verdes $D^+ = 0.0267$ y $D^- = 0.1001$

R matriz de intensidades de azules $D^+ = 0.0344$ y $D^- = 0.1084$

En todos los casos $D^+ \; y \; D^-$son menores que $D_{[0.1,49]} = 0.17128$ por lo tanto decimos que no hay evidencia de que los números no sean uniformes a un nivel de significación de $\alpha = 0.1$.

4.3.3 Prueba de correlación serial sobre la imagen

Aplicado a la sección de la imagen:

(R) R_ =

-0.0003
0.0003
0.0001
0.0005
-0.0002
-0.0007
-0.0016
0.0011
-0.0018

IR_ =

-0.0035	0.0029
-0.0029	0.0035
-0.0031	0.0033
-0.0027	0.0037
-0.0034	0.0030
-0.0039	0.0025
-0.0048	0.0016
-0.0021	0.0043
-0.0050	0.0014

(G) R_ =

0.0022
0.0005
-0.0043
-0.0002
-0.0010
0.0010
0.0008
-0.0003
0.0036

IR_ =

-0.0010	0.0054
-0.0027	0.0037
-0.0075	-0.0011
-0.0034	0.0030
-0.0042	0.0022
-0.0022	0.0042
-0.0024	0.0040
-0.0035	0.0029
0.0004	0.0069

(B) R_ =

-0.0003
0.0013
-0.0027
0.0024
0.0007
-0.0002
-0.0026
-0.0005
-0.0007

IR_ =

-0.0035	0.0029
-0.0019	0.0045
-0.0059	0.0005
-0.0008	0.0057
-0.0025	0.0039
-0.0034	0.0030
-0.0058	0.0006
-0.0037	0.0027
-0.0039	0.0025

Todos los intervalos incluyen el cero y podemos asumir todas las covarianzas como estadísticamente no significativas al nivel de confianza $\alpha = 0.05$ o $Z_{0.975} = 1.958$. Es decir, no rechazamos la hipótesis de que no haya correlación significativa.

4.3.4 Prueba serial sobre la imagen

Aplicado a la sección de la imagen:

Prueba serial u = matvec(XNR,250,300,250,300);

```
k=5
(R)O =  50   50   62   54   40
        57   51   48   45   42
        46   53   64   34   51
        56   51   64   54   57
        59   53   47   54   47
  T = 23.9808
(G) O = 50   53   54   63   44
        58   54   48   56   55
        48   59   43   57   62
        51   45   48   34   56
        48   38   62   58   48
  T = 25.3077
(B) O = 47   41   37   39   46
        65   57   54   45   45
        51   49   65   43   53
        49   56   52   57   48
        54   56   68   54   58
  T = 29.1346
```

T es menor que $\chi^2_{[0.1;24]} = 33.2$, por lo tanto, aceptamos que los números son uniformes en dos dimensiones a un nivel de significación de $\alpha = 0.1$

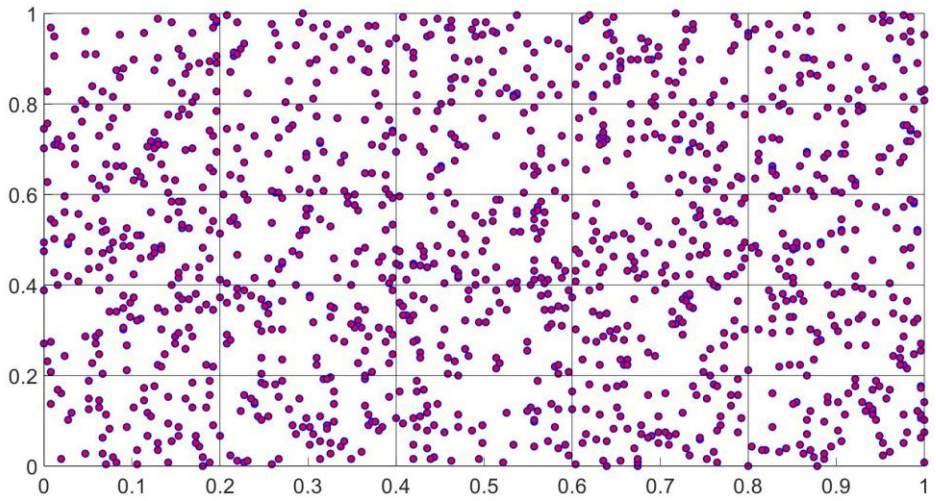

Figura 50. Distribución de pares no solapados de rojos

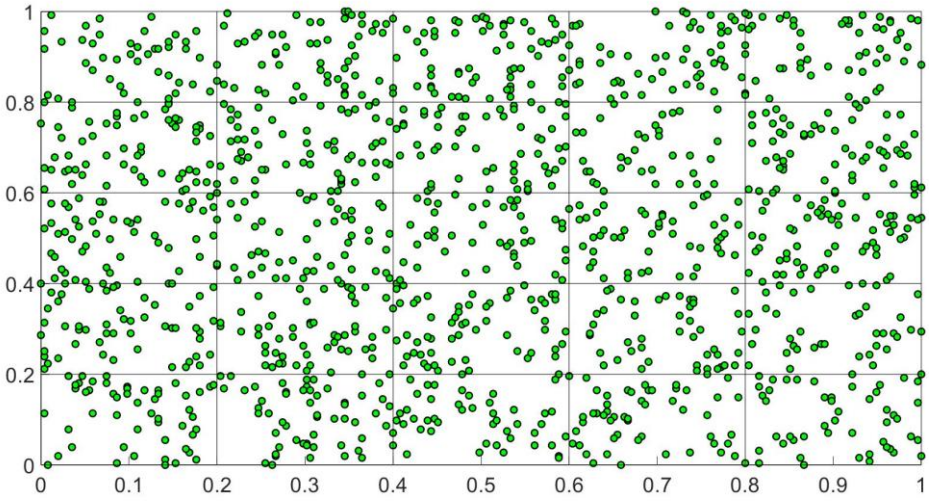

Figura 51. Distribución de pares no solapados de verdes

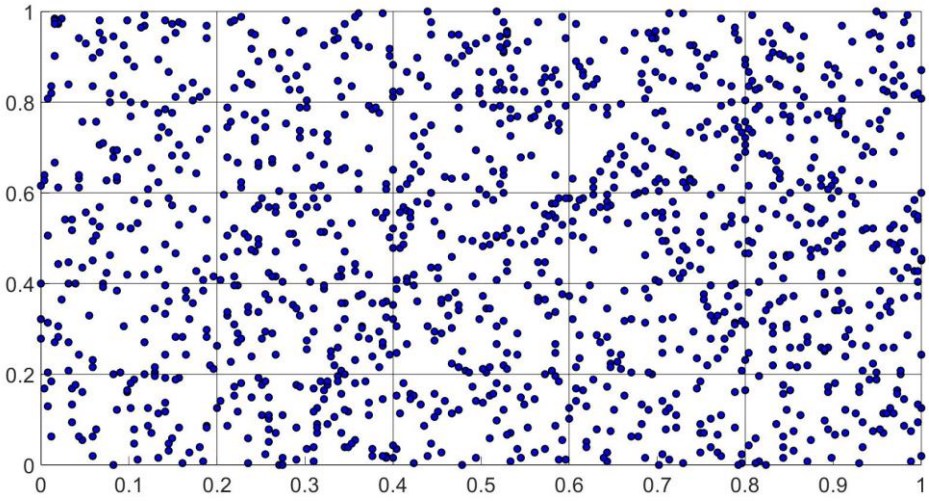

Figura 52. Distribución de pares no solapados de azules

4.3.5 Pruebas de Rachas sobre la imagen

4.3.5.1 Número de rachas crecientes y decrecientes (imagen)

Aplicado a la sección de imagen:

(R) R = 1755	(G) R = 1731	(B) R = 1752
Z0 = 0.9924	Z0 = -0.1241	Z0 = 0.8529
mu = 1.7337e+03	mu = 1.7337e+03	mu = 1.7337e+03
sigma = 21.4960	sigma = 21.4960	sigma = 21.4960

Como $Z_{0,R} = 0.9924, Z_{0,G} = -0.1241, Z_{0,B} = 0.8529$ son menores que $Z_{\alpha/2} = 1.645$, no rechazamos la hipótesis de independencia a un nivel de significación de $\alpha = 0.1$.

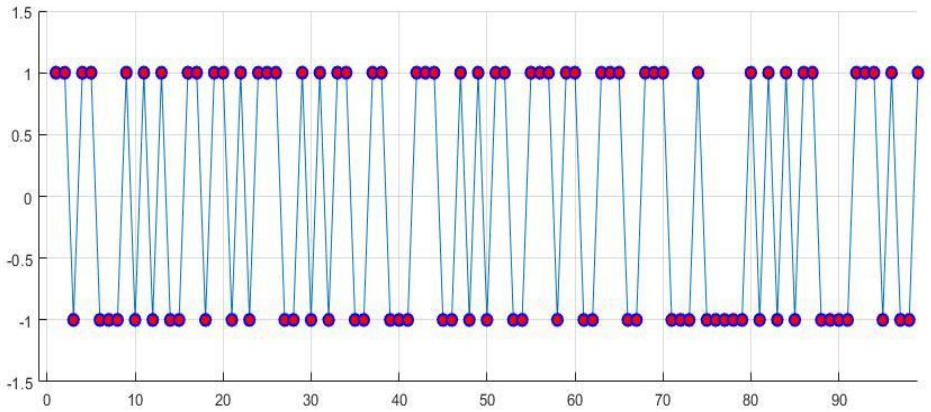

Figura 53. Distribución de las rachas crecientes y decrecientes en la muestra dada por las 100 primeras entradas de la matriz de la sección de imagen

4.3.5.2 Número de rachas por encima y por debajo de la media (imagen)

Aplicado a la sección de la imagen:

(R) R = 1309	(G) R = 1257	(B) R = 1281
Z0 = 0.3320	Z0 = -1.7249	Z0 = -0.7282
mu = 1.3005e+03	mu = 1.3005e+03	mu = 1.3005e+03
sigma = 25.4860	sigma = 25.4860	sigma = 25.4860
n1 = 1276	n1 = 1276	n1 = 1276
n2 = 1325	n2 = 1325	n2 = 1325

Como $Z_{0,R} = 0.3320, Z_{0,G} = -1.7249, Z_{0,B} = -0.7282$ son menores que $Z_{\alpha/2} = 1.645$, no rechazamos la hipótesis de independencia a un nivel de significación de $\alpha = 0.1$.

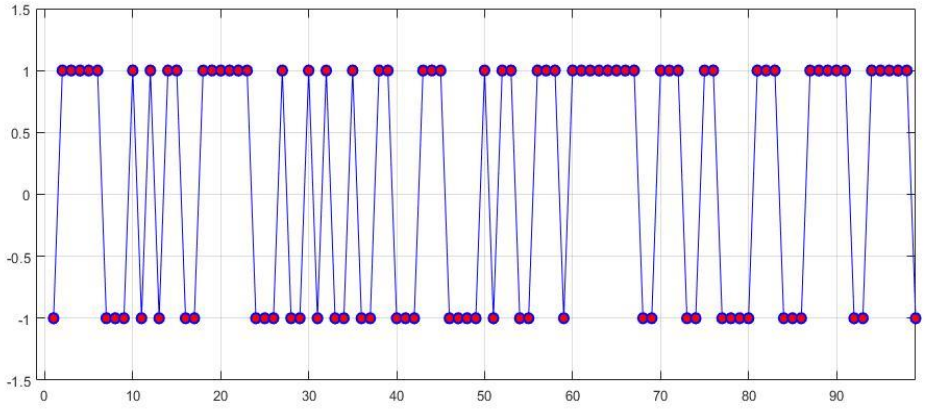

Figura 54. Distribución de las rachas por encima y por debajo de la media en la muestra dada por las 100 primeras entradas de la matriz de la sección de imagen

4.3.5.3 Longitud de rachas crecientes y decrecientes (imagen)

Aplicado a la sección de la imagen:

(R)	chi2 = 2.7453	
	O = 1117	E = 1.0838*1.0e+03
	473	0.4766*1.0e+03
	128	0.1371*1.0e+03
	33	0.0299*1.0e+03
	4	0.0062*1.0e+03
(G)	chi2 = 3.4682	
	O = 1098	E = 1.0838*1.0e+03
	447	0.4766*1.0e+03
	144	0.1371*1.0e+03
	34	0.0062*1.0e+03
	8	0.0299*1.0e+03
(B)	chi2 = 7.6266	
	O = 1123	E = 1.0838*1.0e+03
	444	0.4766*1.0e+03
	156	0.1371*1.0e+03
	24	0.0299*1.0e+03
	5	0.0062*1.0e+03

Como $\chi^2_{0,R} = 2.7453, \chi^2_{0,G} = 3.4682, \chi^2_{0,B} = 7.6266$ son menores que $\chi^2_{[0.1;4]} = 7.7794$, no rechazamos la hipótesis de independencia a un nivel de significación de $\alpha = 0.1$.

4.3.5.4 Longitud de rachas por sobre y bajo la media (imagen)

Aplicado a la sección de la imagen:

(R) chi2 = 3.0389

O =	E =
653	649.5580
340	324.7790
158	162.4471
79	81.2812
38	40.6838
18	20.3707
14	10.2034
5	5.1125
4	5.1414

Como $\chi^2_{0,R} = 3.0389$ es menor que $\chi^2_{[0.1;8]} = 13.3616$, no rechazamos la hipótesis de independencia a un nivel de significación de $\alpha = 0.1$.

(G) chi2 = 14.0866

O =	E =
610	650.2151
308	325.1076
175	162.5567
72	81.2812
50	40.6428
16	20.3229
9	10.1623
7	5.0817
10	5.0832

Como $\chi^2_{0,G} = 14.0866$ es menor que $\chi^2_{[0.05;8]} = 15.5073$, no rechazamos la hipótesis de independencia a un nivel de significación de $\alpha = 0.1$.

(B) chi2 = 4.1719

O =	E =
631	648.0707
316	324.0354
175	162.1989
73	81.2807
41	40.7764
22	20.4789
10	10.2962
8	5.1822
5	5.2737

Como $\chi^2_{0,B} = 4.1719$ es menor que $\chi^2_{[0.1;8]} = 13.3616$, no rechazamos la hipótesis de independencia a un nivel de significación de $\alpha = 0.1$.

4.4 Análisis del rendimiento temporal

Como sucede con los sistemas tecnológicos en general, la calidad del resultado es proporcional al trabajo invertido, en este caso, trabajo computacional, por tanto, con potencias computacionales iguales (como es el caso normal ya que normalmente utilizaremos el mismo procesador para todos los programas) será mayor el tiempo de cálculo. `Codefoto1`, proporciona más calidad de encriptación, pero con más tiempo de cálculo, lo cual es lógico si recordamos que recalcula las matrices pseudoaleatorias en cada iteración, a diferencia de los otros programas que hacen ese cálculo una sola vez y utilizan siempre la misma. Por tanto, la conclusión que sacamos es que utilizaremos `codefoto1` para lograr codificaciones de mucha calidad sin restricciones en el tipo de imagen, `codefoto3` se aplicará en imágenes para las que se requiera una codificación menos robusta, mientras que `codefoto2` será aplicado únicamente a imágenes irregulares, obteniendo la codificación de menor calidad, pero suficiente. Lo mismo ocurre por definición con sus respectivos programas de decodificación.

Más allá de una interpretación cualitativa de la relación de calidad de codificación y tiempo de operación, es interesante comprobar matemáticamente esta relación. Para ello ejecutaremos los programas `codefoto1`, `codefoto2` y `codefoto3` sobre una imagen escalada a distintos tamaños, con las funciones `tic` y `toc`, lo que nos permitirá medir el tiempo de ejecución de cada prueba con imágenes de distinto tamaño para cada programa, y con la información obtenida podemos trazar gráficas de mejor ajuste para estimar el tiempo requerido de cálculo para cada tamaño de imagen.

Utilizaremos para ello la imagen de un paisaje montañoso que vemos a continuación:

Figura 55. Imagen de paisaje

Es una imagen rectangular, no cuadrada, por lo tanto, normalizaremos las imágenes, clasificándolas por su área, más concretamente con la raíz del área, como si se tratase de imágenes cuadradas de lado igual a este valor, de forma que la relación con el tiempo de cálculo sea generalizable para cualquier foto de cualquier proporción. Esta suposición es correcta si asumimos que el tiempo de cifrado es igual para dos imágenes de distintas dimensiones pero igual superficie, lo cual es asumible ya que ello implica igual número de píxeles, y ya que el algoritmo realiza tantas operaciones relativas a la cantidad, es una suposición correcta, la única diferencia son las operaciones de más que realiza para transformar los píxeles de la foto más estrecha en un vector columna, que podemos suponer despreciables frente al tiempo absoluto de ejecución.

A continuación, representamos la tabla de tiempos de ejecución de cada programa, y tres gráficas, una para cada sistema de cifrado y descifrado, y otras dos, una con todos los métodos de cifrado y la otra de descifrado, obtenidas con un portátil MSI PE72 8RD (procesador Intel Core i7-8750H CPU 2.20GHZ, con 16 GB de RAM y Windows10 de 64 bits) y Matlab R2020A:

	Tiempos (s)					
Nº pixeles verticales	Codefoto1	Decodefoto1	Codefoto2	Decodefoto2	Codefoto3	Decodefoto3
100	1.0841	1.17467	0.0831	0.0872	0.1507	0.0620
200	4.3526	7.03221	0.2229	0.2491	0.2432	0.2208
300	9.2152	14.8740	0.4820	0.5064	0.5229	0.4884
400	16.5467	26.7737	0.8517	0.9060	0.9373	0.8688
500	25.8038	41.6816	1.3407	1.4043	1.5195	1.3790
600	37.2172	60.5432	1.9282	2.0704	2.2608	1.9641
700	50.3636	81.3575	2.5956	2.7086	3.2100	2.8647
800	66.4255	108.3042	3.4195	3.5531	4.4938	3.5823
900	83.3701	133.7348	4.3111	4.5424	5.9327	4.5021
1000	104.1700	167.9602	5.3877	5.5168	7.4981	5.6187
1100	125.0873	202.6555	6.4564	6.7633	9.3202	6.7090
1200	149.9251	243.7716	7.6308	7.9836	11.5533	8.1194
1300	176.0121	284.7327	8.9189	9.4360	14.2727	9.3831
1400	203.1236	329.2001	10.4991	10.9265	16.7026	11.0526
1500	234.5564	378.9953	11.8998	12.5478	20.1104	12.8849
1600	272.9992	442.3498	13.7632	14.4560	23.5510	15.0645
1700	309.8477	501.3712	15.2430	16.2536	26.9435	16.3568
1800	382.3542	597.7224	17.1376	18.5528	31.4948	18.9721
1900	404.0264	653.3349	19.1314	20.5429	35.6560	21.4749
2000	434.2794	700.3451	21.1885	22.5710	40.4380	23.9316
2100	484.8653	785.6923	23.7241	25.0178	46.3051	26.4180
2200	539.9464	872.3309	29.7873	28.1791	51.7158	28.2523
2300	607.1211	981.1001	31.7547	30.2885	57.4900	31.8825
2400	634.2153	1023.6778	34.4910	33.0819	63.8277	33.1355
2500	705.5113	1143.3077	35.2436	35.9263	73.4996	34.7520
2592	744.9982	1210.7644	36.4763	39.8586	80.0643	39.4398

Tabla 1: Tiempos de operación de los programas

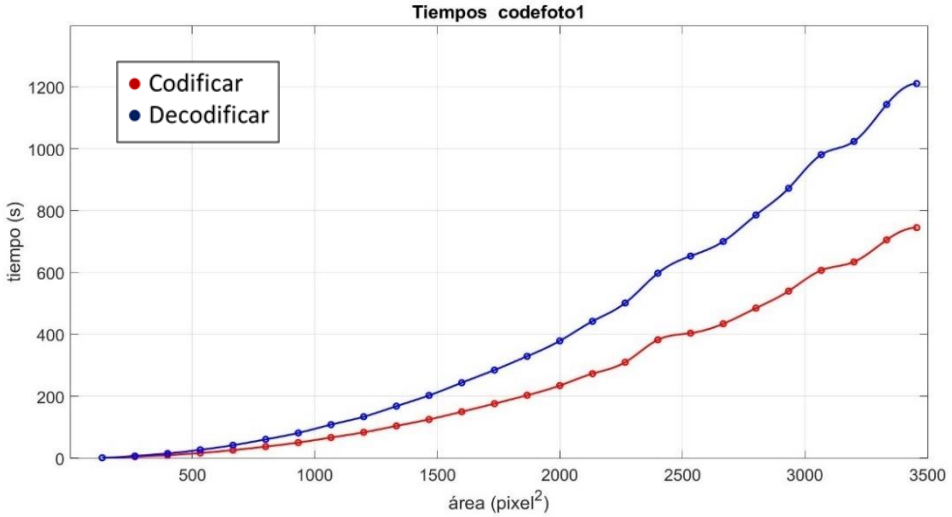

Figura 56. Tiempos de cifrado y descifrado de codefoto1

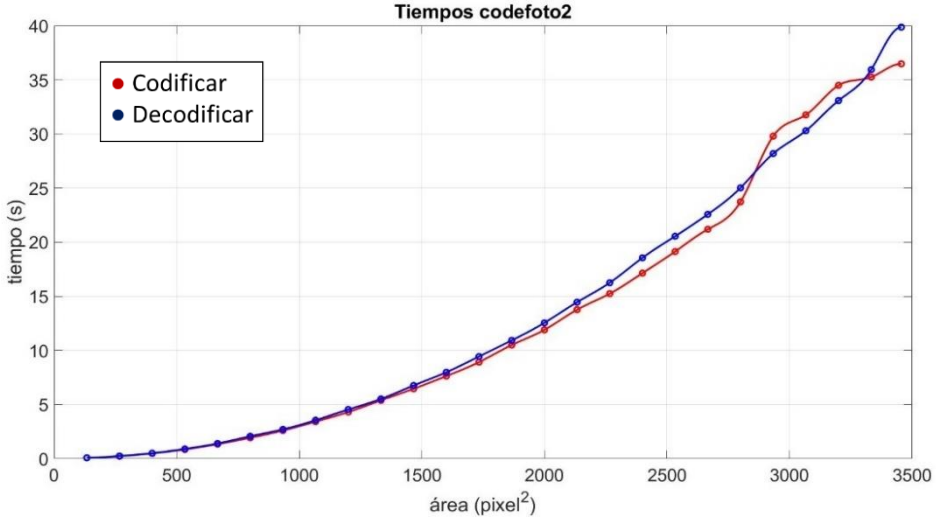

Figura 57. Tiempos de cifrado y descifrado de codefoto2

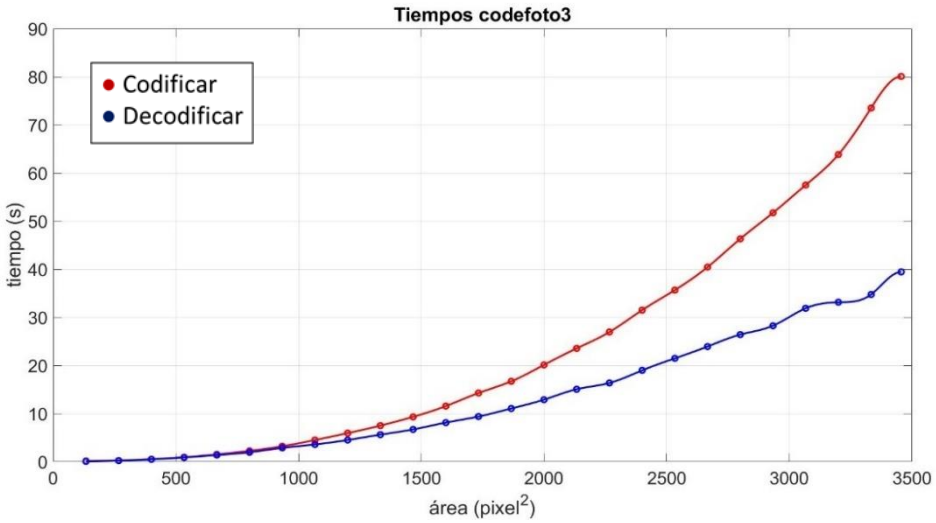

Figura 58. Tiempos de cifrado y descifrado de codefoto3

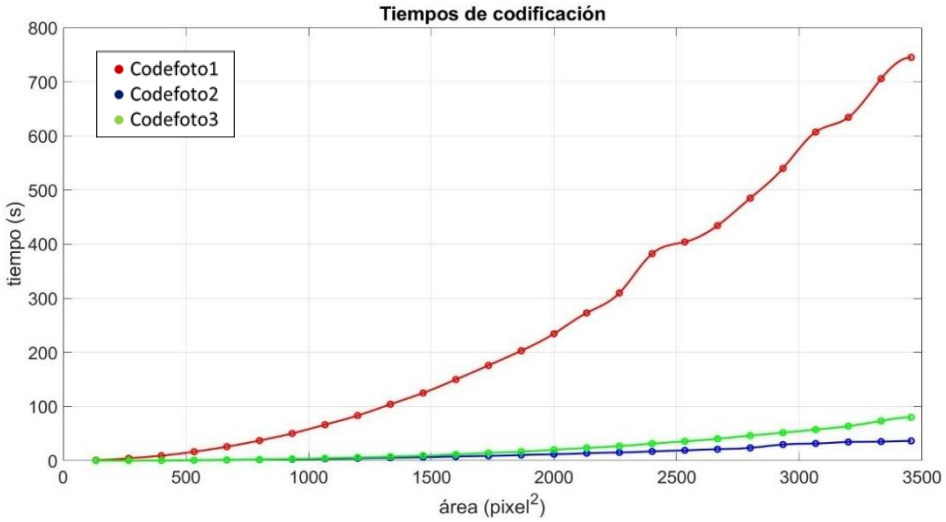

Figura 59. Tiempos de cifrado de los tres sistemas

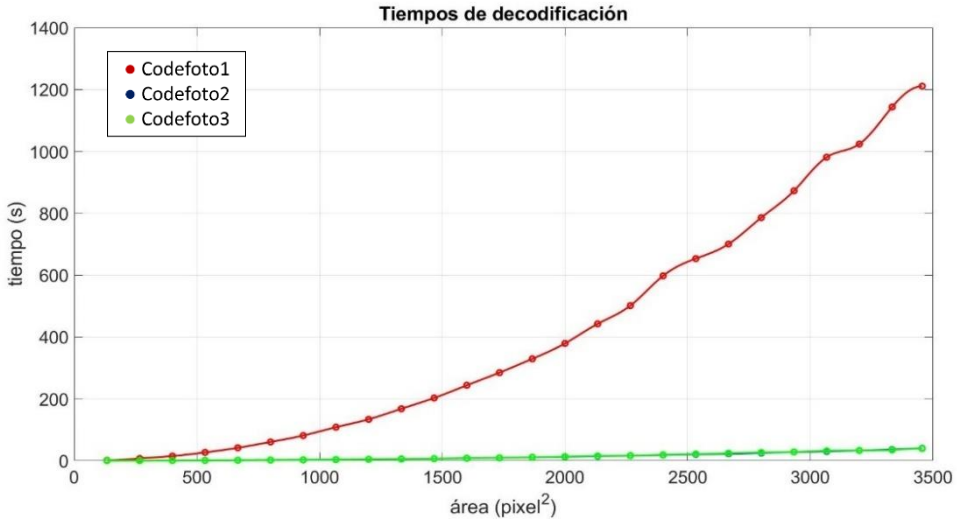

Figura 60. Tiempos de descifrado de los tres sistemas

A continuación, se hablará de *codefoto1*, *codefoto2* y *codefoto3* para referirse a los sistemas de codificación que engloban respectivamente a los programas `codefoto1` y `decodefoto1`, `codefoto2` y `decodefoto2` y `codefoto3` y `decodefoto3`.

Se pueden sacar varias conclusiones de esta información, en primer lugar, vemos que el tiempo de decodificación de *codefoto1* es mayor que el de codificación, mientras que con *codefoto3* pasa lo contrario y en *codefoto2* los tiempos están prácticamente solapados. Pero lo más relevante es que, tanto para codificación como para decodificación, el sistema *codefoto1* es con diferencia el que más tiempo requiere con todo tamaño de imágenes, mostrando una evolución aparentemente exponencial, aunque de razón pequeña, del tiempo conforme aumenta el tamaño de la imagen, mientras que los otros dos sistemas tienen alcanzan tiempos mucho inferiores, con *codefoto3* tardando aproximadamente el doble en codificación para muchos tamaños, mientras que en decodificación el rendimiento temporal de ambos prácticamente se solapa.

La diferencia con los otros algoritmos es considerable, tanto en codificación como en decodificación, para la imagen más grande, de área 3456^2 píxeles, el tiempo que tarda *codefoto1* en codificar es de 755 segundos o 12 minutos con 35 segundos, lo que es: 744.9982/80.0643=9.3045 veces mayor que el siguiente que más tarda que es *codefoto3*; mientras que en decodificación, para la imagen más grande tarda 1210.7644 segundos siendo: 1210.7644/39.8586=30.3573 veces más lento que *codefoto3*, y prácticamente igual con *codefoto2*. En definitiva, es mucho más eficiente para codificar fotos muy grandes utilizar *codefoto2* y *codefoto3*, preferiblemente *codefoto3*, ya que, por definición, ofrece una codificación más fiable; siempre y cuando las imágenes no tengan secciones considerables de colores regulares o en patrones periódicos, en cuyo caso es necesario utilizar *codefoto1*. Con procesadores muy potentes es recomendable utilizar siempre *codefoto1* ya que es sin duda la que ofrece una codificación más fiable.

4.4.1 Caltime

Hemos programado una función de Matlab llamada `caltime` que permite estimar el tiempo de codificación y decodificación para cualquier ordenador y versión de Matlab. Presenta la forma `[x,tc,td] = caltime(metodo,factor,X_,Y_)` donde `método` toma los valores 1,2 y 3 según se estime el tiempo para *codefoto1*, *codefoto2* y *codefoto3* respectivamente, `factor` es un número que representa la velocidad del procesador y de la versión de Matlab empleada y `X_` e `Y_` son el número de pixeles horizontales y verticales de la foto. El valor que toma factor se calcula de la siguiente forma: para una foto de tamaño pequeño se calcula el tiempo de codificación `tc1`, después se aplica `caltime` tomando `factor=1` para calcular `tc` y finalmente para el resto de las imágenes se toma `factor=tc1/tc`. El código está expuesto en el anexo B.

5. Conclusión

El cifrado de Hill con generación de matrices pseudoaleatorias es, en definitiva, un método de cifrado muy robusto y seguro, en especial cuando se generan nuevas matrices iterando el algoritmo generador tras cada operación de multiplicación matricial del cálculo del cifrado. Este método, llamado sistema *codefoto1*, es con diferencia el más fiable, todas las imágenes que codifica son indescifrables e irreconocibles de la original a simple vista. El método más simple, que llamamos *codefoto2*, con el uso de un solo par de matrices pseudoaleatorias generadas con una única iteración del generador, no ofrece un cifrado seguro para imágenes con regiones de colores invariantes o con patrones regulares, ya que la imagen cifrada muestra contornos de la imagen anterior, hasta el punto de que es posible descifrar números que había en la imagen en claro. Para mejorar esta situación, se puede introducir una etapa de permutación de píxeles que mejora la aleatoriedad en la imagen codificada, el resultado es un algoritmo llamado *codefoto3* que sigue fallando en imágenes regulares o en patrones periódicos, pero que ofrece mejor encriptación de la imagen, ya que, aun siendo reconocibles patrones en el cifrado de estas imágenes, con el nuevo método no es tan evidente estimar la imagen previa el cifrado.

Las pruebas estadísticas realizadas demuestran, con un corto margen de error, la uniformidad y aleatoriedad del algoritmo, tanto del generador de matrices pseudoaleatorias, que es, en definitiva, la pieza fundamental para lograr un cifrado de Hill muy fiable, cómo de las codificaciones obtenidas por el primer método de cifrado, lo que avala su eficacia que ya se ve a simple vista. Sin embargo, la contrapartida del método más eficaz es que es además el más lento, ya que requiere muchas más operaciones que para procesadores de alcance común hacen el proceso muy tedioso, que aumenta exponencialmente con el tamaño de la imagen, para imágenes de 3456^2 píxeles el tiempo de ejecución es de 755 segundos, el segundo y tercer método en cambio, ocupan tiempos similares y muy inferiores al primero, para imágenes iguales. Por tanto, el primer método se utilizará con toda clase de imágenes teniendo en cuenta el coste temporal que supone, mientras que los otros dos, preferiblemente el tercero, se usaran en imágenes irregulares, normalmente de gran tamaño, donde podamos sacrificar fiabilidad de cifrado por velocidad. Por medio del algoritmo de Diffie-Hellman implementado a Matlab hemos logrado realizar una codificación en clave pública de la imagen. Esto nos permite hacer mucho más seguro el envío de claves por medios donde pueden ser interceptadas, ya que para descifrar la imagen el hacker tiene además que afrontar un cálculo muy costoso de calcular computacionalmente por fuerza bruta, el problema del logaritmo discreto. Además, el algoritmo de Diffie-Hellman puede ser usado para cifrar una, más de una, o todas las variables de la clave, haciendo exponencialmente más difícil descifrar la clave completa.

Finalmente, el sistema de cifrado al completo se ha implementado en dos aplicaciones de MATLAB, que posteriormente hemos convertido en ejecutables (ver figuras 61-68) y en versiones web (ver figuras 69 y 70) que se encuentran en el repositorio Riunet de la Universitat Politècnica de València [12,13]. En este formato, el programa es muy sencillo de utilizar por un usuario inexperto ya que las interfaces de usuario son muy interactivas y entendibles, y además cuenta con un texto desplegable de ayuda que contiene las instrucciones de uso explicadas muy didácticamente para que el usuario pueda hacer uso de la herramienta de inmediato.

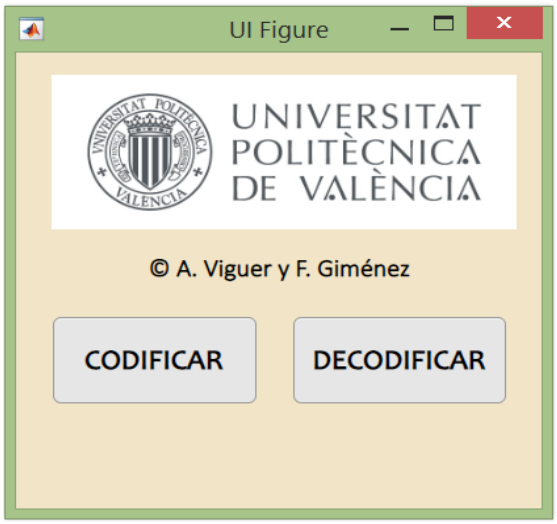

Figura 61. Inicio del laboratorio virtual CODE y DECODE (versión ejecutable)

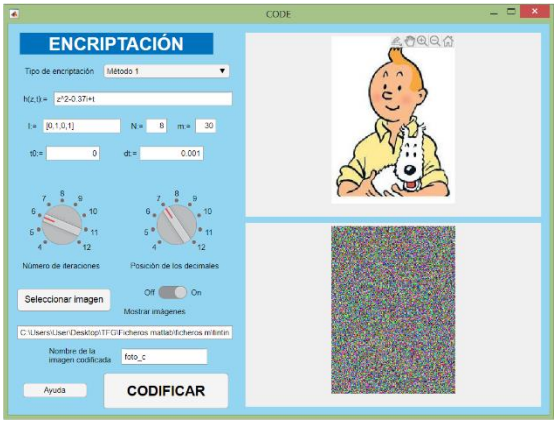

Figura 62. Laboratorio virtual CODE (versión ejecutable)

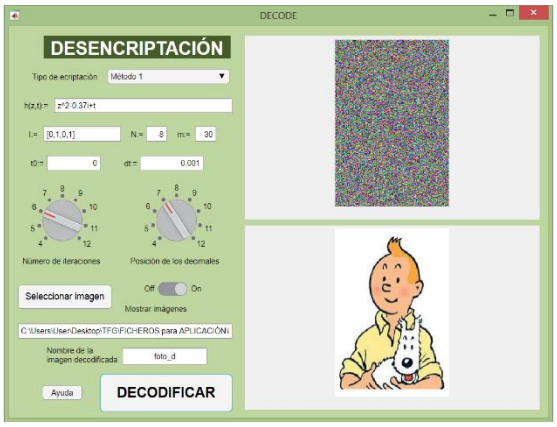

Figura 63. Laboratorio virtual DECODE (versión ejecutable)

Figura 64. Inicio del laboratorio virtual Diffie-Hellman (versión ejecutable)

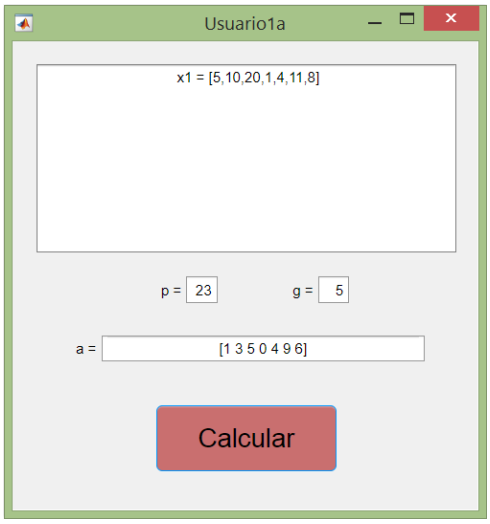

Figura 65. Programa Usuario1a del Laboratorio virtual Diffie-Hellman (versión ejecutable)

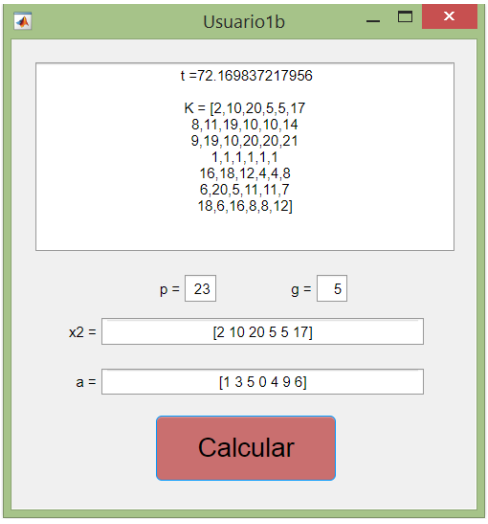

Figura 66. Programa Usuario1b del Laboratorio virtual Diffie-Hellman (versión ejecutable)

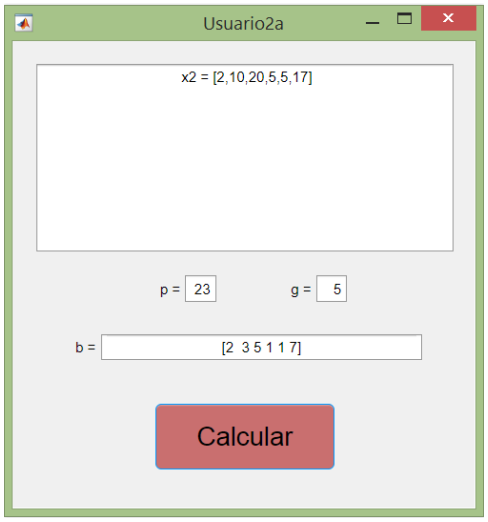

Figura 67. Programa Usuario2a del Laboratorio virtual Diffie-Hellman (versión ejecutable)

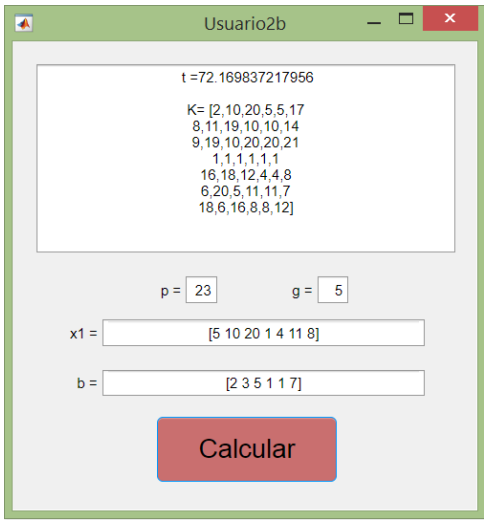

Figura 68. Programa Usuario2b del Laboratorio virtual Diffie-Hellman (versión ejecutable)

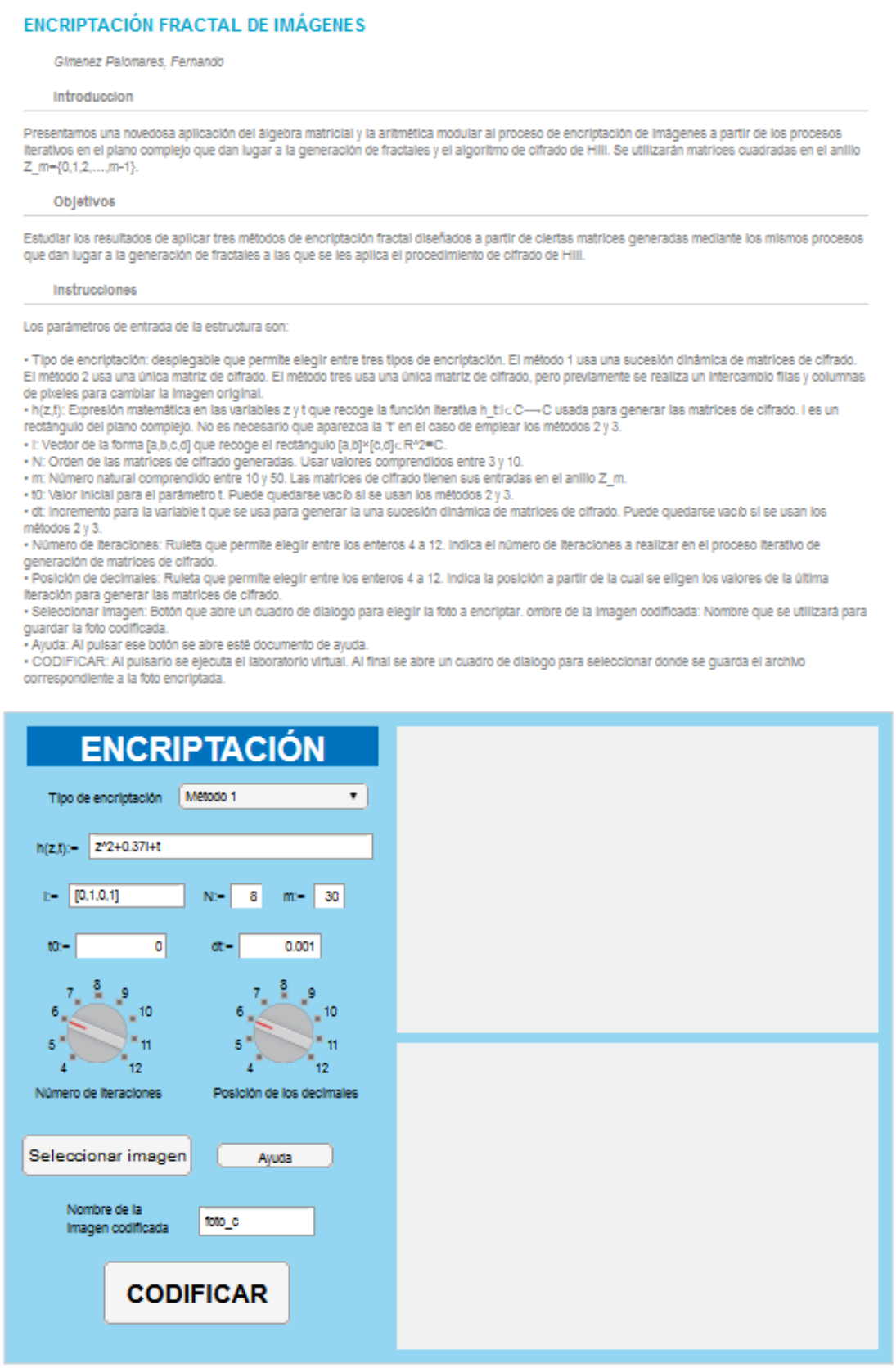

Figura 69. Laboratorio virtual CODE (versión web)

DESENCRIPTACIÓN FRACTAL DE IMÁGENES

Gimenez Palomares, Fernando

Introduccion

Presentamos una novedosa aplicación del álgebra matricial y la aritmética modular al proceso de encriptación de imágenes a partir de los procesos iterativos en el plano complejo que dan lugar a la generación de fractales y el algoritmo de cifrado de Hill. Se utilizarán matrices cuadradas en el anillo Z_m={0,1,2,....,m-1}.

Objetivos

Estudiar los resultados de aplicar tres métodos de encriptación fractal diseñados a partir de ciertas matrices generadas mediante los mismos procesos que dan lugar a la generación de fractales a las que se les aplica el procedimiento de cifrado de Hill.

Instrucciones

Los parámetros de entrada de la estructura son:

- Tipo de encriptación: desplegable que permite elegir entre tres tipos de encriptación. El método 1 usa una sucesión dinámica de matrices de cifrado. El método 2 usa una única matriz de cifrado. El método tres usa una única matriz de cifrado, pero previamente se realiza un intercambio filas y columnas de pixeles para cambiar la imagen original.
- h(z,t): Expresión matemática en las variables z y t que recoge la función iterativa $h_t : C \longrightarrow C$ usada para generar las matrices de cifrado. I es un rectángulo del plano complejo. No es necesario que aparezca la 't' en el caso de emplear los métodos 2 y 3.
- I: Vector de la forma [a,b,c,d] que recoge el rectángulo $[a,b]\times[c,d] \subset R^2 = C$.
- N: Orden de las matrices de cifrado generadas. Usar valores comprendidos entre 3 y 10.
- m: Número natural comprendido entre 10 y 50. Las matrices de cifrado tienen sus entradas en el anillo Z_m.
- t0: Valor inicial para el parámetro t. Puede quedarse vacío si se usan los métodos 2 y 3.
- dt: Incremento para la variable t que se usa para generar la una sucesión dinámica de matrices de cifrado. Puede quedarse vacío si se usan los métodos 2 y 3.
- Número de iteraciones: Ruleta que permite elegir entre los enteros 4 a 12. Indica el número de iteraciones a realizar en el proceso iterativo de generación de matrices de cifrado.
- Posición de decimales: Ruleta que permite elegir entre los enteros 4 a 12. Indica la posición a partir de la cual se eligen los valores de la última iteración para generar las matrices de cifrado.
- Seleccionar imagen: Botón que abre un cuadro de diálogo para elegir la foto a desencriptar.
- Nombre de la imagen decodificada: Nombre que se utilizará para guardar la foto DECODIFICADA.
- Ayuda: Al pulsar ese botón se abre este documento de ayuda.
- DECODIFICAR: Al pulsarlo se ejecuta el laboratorio virtual. Al final se abre un cuadro de diálogo para seleccionar donde se guarda el archivo correspondiente a la foto desencriptada.

Figura 70. Laboratorio virtual DECODE (versión web)

5.1 Nuevas líneas de investigación

Los programas de codificación diseñados en el proyecto son muy satisfactorios, pero tan solo abren las puertas a todas las posibilidades que ofrece la codificación programada en el entorno Matlab. Un primer paso es explorar nuevos métodos de cifrado, tanto para complementar y añadir al programa base de cifrado de Hill como para integrar un algoritmo nuevo. Una posibilidad muy atractiva es implementar el algoritmo Rijndael (también conocido como *Advanced Encryption Standard*) para codificar imágenes. Otra posibilidad es expandir las operaciones de multiplicación y suma matricial del cifrado de Hill, agregando operaciones nuevas de suma y producto. Si el cifrado original es:

$$\text{codificación:} \qquad \tilde{y}_j = A_j \tilde{x}_j + v_j \quad (\text{módulo } m) \qquad [2.11]$$

$$\text{decodificación:} \qquad \tilde{x}_j = A^{-1}(\tilde{y}_1 - v) \quad (\text{módulo } m) \qquad [2.12]$$

Podemos iterar el generador de matrices pseudoaleatorias dos veces en cada operación de cifrado para así obtener una matriz y un vector pseudoaleatorios adicionales e implementar el cifrado de la siguiente forma:

$$\text{codificación:} \qquad \tilde{y}_j = A_{j,n+1}\left(A_{j,n}\tilde{x}_j + v_{j,n}\right) + v_{j,n+1} \text{ (módulo } m) \qquad [5.1]$$

$$\text{decodificación:} \qquad \tilde{x}_j = A_{j,n+1}^{-1}\left(A_{j,n}^{-1}\left(\tilde{y}_j - v_{j,n}\right) - v_{j,n+1}\right) \quad (\text{módulo } m) \qquad [5.2]$$

siendo n la iteración actual a la entrada del nuevo tramo de la imagen en el cifrado de Hill se puede comprobar que n=2j. $A_{j,n+1}$ y $v_{j,n+1}$ son generados por una iteración más del generador, la $n+1$, la siguiente a n la que genera $A_{j,n}$ y $v_{j,n}$. De hecho, podríamos implementar esto un número indefinido de veces, aumentando la complejidad de la encriptación, pero también su coste computacional.

Por otro lado, otra línea de expansión es introducir cifrado de texto, para cifrar los nombres de los archivos. Es una opción que se consideró introducir en este proyecto, sin embargo, para enfocar el proyecto únicamente al cifrado de imágenes, y dado que el cifrado de texto es la opción de codificación más estándar por así decir, al final no se incluyó, centrando el trabajo a las imágenes. Cifrar además el nombre de la imagen permitiría seguridad total contra intercepción o espionaje ya que el interceptor no podría conocer ni el nombre del archivo envía, más allá de no poder descifrar la imagen. Se pueden utilizar perfectamente los algoritmos de cifrado de Hill utilizados, *codefoto1*, *codefoto2* y *codefoto3*, ajustándolos al cifrado de texto, lo cual sería realmente simplificarlos, e incluso utilizar el algoritmo de Diffie-Hellman también para conseguir el mismo resultado que con las imágenes. O se pueden utilizar nuevos algoritmos de cifrado o variaciones del cifrado de Hill, las posibilidades son infinitas. Con Matlab, trasponer caracteres a valores numéricos para operar sobre estos es muy sencillo, con el comando `UnicodeValues` podemos pasar caracteres a valores, y a la inversa, con la función `char` podemos pasar valores numéricos a caracteres.

Finalmente, las opciones más desafiantes de expansión del sistema serían las que pasan por automatizar el cifrado de forma que se pueda implementar para cifrar grandes lotes de imágenes en un solo proceso, es decir, en un solo clic. Más allá de esto, si además se implantase un cifrado de sonido se podría llevar a cabo el cifrado multimedia, de vídeos, o bien imágenes sueltas o sonido, ya que, al fin y al cabo, un vídeo es un conjunto de fotogramas sincronizados con un sonido, si podemos cifrar el conjunto de fotogramas por un lado y el sonido por el otro podemos codificar el vídeo.

6. Anexos

6.1 Anexo A: Programación de las pruebas estadísticas

- Código prueba Chi-cuadrado

```
function T = test_chi(V,k)
maxV=max(V);
Vnorm=V/maxV;
Vs=sort(Vnorm);
n=numel(Vs);
T=0;
O = zeros(k,1);
for j=1:k
    cont = 0;
    for z=1:n
        if (Vs(z)>(j-1)/k)&&(Vs(z)<=j/k)
            cont = cont+1;
        end
    end
    O(j) = cont;
    T=T+((O(j)-n/k)^2)/(n/k);
end
histogram(O)
```

- Código prueba de Kolmogorov-Smirnov

```
function [Dmas,Dmenos] = test_KS(u)
u = sort(u);
n = numel(u);
J = (1:n)';
v1 = J/n-u;
v2 = u-(J-1)/n;
Dmas = max(v1);
Dmenos = max(v2);
```

- Código prueba de correlación serial

```
function [R,IR] = test_CS(u,k,Z_alfa)
n=numel(u);
R=0;
for i=1:n-k
    R=R+(u(i)-0.5)*(u(i+k)-0.5)/(n-k);
end
IR = [R-Z_alfa/(12*sqrt(n-k)),R+Z_alfa/(12*sqrt(n-k))];
```

- Código prueba serial

```
function [T,O] = test_pserial(u,k)
n = numel(u);
if mod(n,2)>0 % si n es impar
    u = u(1:n-1);
end
n = numel(u);
u = u/max(u);
% Gráfico
figure(3)
hold on
for s=1:2:n-1
%       plot(u(s),u(s+1),'bx')
    plot(u(s),u(s+1),'go',...
        'LineWidth',1.5,...
        'MarkerSize',7,...
        'MarkerEdgeColor','k',...
        'MarkerFaceColor','g')
end
for s=0:k
    plot([s/k s/k],[0 1],'k')
    plot([0 1],[s/k s/k],'k')
end
hold off
y = u;
for i=1:n
    for j=1:k
        if (u(i)>(j-1)/k)&&(u(i)<=j/k)
            y(i) = j;
        end
    end
end
O = zeros(k);
for i=1:k
    for j=1:k
        t = 0;
        for s=1:2:n-1
            if (y(s)==i)&&(y(s+1)==j)
                t = t+1;
            end
        end
        O(i,j) = t;
    end
end
T = 0;
for i=1:k
    for j=1:k
        T = T+(O(i,j)-n/(2*k^2))^2/(n/(2*k^2));
    end
end
```

- Código de las pruebas de rachas

- Código número de rachas crecientes y decrecientes

```
function [R,Z0,mu,sigma,y] = test_RachasCN(u)
y = [];
n = numel(u);
for i=1:n-1
    if u(i)<=u(i+1)
        y = [y 1];
    else
        y = [y -1];
    end
end
hold on
plot(y)
plot(y,'go',...
        'LineWidth',1.5,...
        'MarkerSize',7,...
        'MarkerEdgeColor','b',...
        'MarkerFaceColor','r')
    axis([-1,n-1,-1.5,1.5])
    hold off
R = 1;
for i=1:n-2
    if sign(y(i)*y(i+1))==-1
        R = R+1;
    end
end
mu = (2*n-1)/3;
sigma = sqrt((16*n-29)/90);
 Z0 = (R-mu)/sigma;
```

- Código número de rachas por encima y por debajo de la media

```
function [R,Z0,mu,sigma,n1,n2] = test_RachasMN(u)
y = [];
n = numel(u);
for i=1:n
    if u(i)<=0.5
        y = [y -1];
    else
        y = [y 1];
    end
end
hold on
plot(y)
plot(y,'go',...
        'LineWidth',1.5,...
        'MarkerSize',7,...
        'MarkerEdgeColor','b',...
        'MarkerFaceColor','r')
axis([-1,n+1,-1.5,1.5])
hold off
R = 1;
for i=1:n-1
    if sign(y(i)*y(i+1))==-1
```

```
        R = R+1;
    end
end
n1 = 0;
for i=1:n
    if y(i)==1
        n1 = n1+1;
    end
end
n2 =n-n1;
mu = 2*n1*n2/n+0.5;
sigma = sqrt(2*n1*n2*(2*n1*n2-n)/(n^2*(n-1)));
 Z0 = (R-mu)/sigma;
```

- **Código longitud de rachas crecientes y decrecientes**

```
function [chi2,O,E] = test_RachasCL(u)
y = [];
n = numel(u);
for i=1:n-1
    if u(i)<=u(i+1)
        y = [y 1];
    else
        y = [y -1];
    end
end
% Contamos rachas

O = zeros(n-1,1);
for k=1:n-2
    s = 0;
    for i=2:n-k-1
        if (norm(y(i:i+k-1)-ones(1,k))<1e-6)&&(y(i-1)==-1)&&(y(i+k)==-1)
            s = s+1;
        end
        if (norm(y(i:i+k-1)+ones(1,k))<1e-6)&&(y(i-1)==1)&&(y(i+k)==1)
            s = s+1;
        end
    end
    if (norm(y(1:k)-ones(1,k))<1e-6)&&(y(1+k)==-1)
        s = s+1;
    end
    if (norm(y(1:k)+ones(1,k))<1e-6)&&(y(1+k)==1)
        s = s+1;
    end
    if (norm(y(n-k:n-1)-ones(1,k))<1e-6)&&(y(n-k-1)==-1)
        s = s+1;
    end
    if (norm(y(n-k:n-1)+ones(1,k))<1e-6)&&(y(n-k-1)==1)
        s = s+1;
    end
    O(k)=s;
end
if norm(y-ones(n-1,1))<1e-6;
    O(n-1)=1;
end
if norm(y+ones(n-1,1))<1e-6;
    O(n-1)=1;
```

```
end
E = zeros(n-1,1);
for i=1:n-2
    E(i) = 2/factorial(i+3)*(n*(i^2+3*i+1)-(i^3+3*i^2-i-4));
end
E(n-1) = 2/factorial(n);
for i=1:n-1
    if E(i)<5
        I = i;
        break
    end
end
E = [E(1:I-2);sum(E(I-1:n-1))];
O = [O(1:I-2);sum(O(I-1:n-1))];
chi2 = sum((O-E).^2./E);
```

- **Código longitud de rachas por encima y por debajo de la media**

```
function [chi2,O,E] = test_RachasML(u)
y = [];
n = numel(u);
for i=1:n
    if u(i)<=0.5
        y = [y -1];
    else
        y = [y 1];
    end
end
% Contamos rachas
n1 = 0;
for i=1:n
    if y(i)==1
        n1 = n1+1;
    end
end
n2 =n-n1;
% Generamos el vector O
O = zeros(n,1);
for k=1:n-2
    s = 0;
    for i=2:n-k
        if (norm(y(i:i+k-1)-ones(1,k))<1e-6)&&(y(i-1)==-1)&&(y(i+k)==-1)
            s = s+1;
        end
        if (norm(y(i:i+k-1)+ones(1,k))<1e-6)&&(y(i-1)==1)&&(y(i+k)==1)
            s = s+1;
        end
    end
    if (norm(y(1:k)-ones(1,k))<1e-6)&&(y(1+k)==-1)
        s = s+1;
    end
    if (norm(y(1:k)+ones(1,k))<1e-6)&&(y(1+k)==1)
        s = s+1;
    end
    if (norm(y(n-k+1:n)-ones(1,k))<1e-6)&&(y(n-k)==-1)
        s = s+1;
    end
    if (norm(y(n-k+1:n)+ones(1,k))<1e-6)&&(y(n-k)==1)
```

```
        s = s+1;
    end
    O(k)=s;
end
s = 0;
if norm(y(1:n-1)-ones(n-1,1))<1e-6;
    s=s+1;
end
if norm(y(2:n)-ones(n-1,1))<1e-6;
    s=s+1;
end
O(n-1)= s;
if norm(y-ones(n,1))<1e-6;
    O(n)=1;
end
if norm(y+ones(n-1,1))<1e-6;
    O(n)=1;
end
E = zeros(n,1);
for i=1:n
    w = (n1/n)^i*(n2/n)+(n1/n)*(n2/n)^i;
    E(i) = n*w/(n1/n2+n2/n1);
end

for i=1:n
    if E(i)<5
        I = i;
        break
    end
end
E = [E(1:I-1);sum(E(I:n))];
O = [O(1:I-1);sum(O(I:n))];
chi2 = sum((O-E).^2./E);
```

6.2 Anexo B: Código de las funciones Matvec y Caltime

- Código de Matvec de transformación de matrices a vectores columna normalizados

```
function u = matvec(X,i1,i2,j1,j2)
u = [];
for i=i1:i2
    for j=j1:j2
        u = [u;X(i,j)];
    end
end
u = u/max(u);
```

- Código de Caltime para el cálculo de tiempos de codificación y decodificación.

```
function [x,tc,td] = caltime(metodo,factor,X_,Y_)

%codefoto 1
T1 = [1.0841
    4.3526
    9.2152
   16.5467
   25.8038
   37.2172
   50.3636
   66.4255
   83.3701
  104.1700
  125.0873
  149.9251
  176.0121
  203.1236
  234.5564
  272.9992
  309.8477
  382.3542
  404.0264
  434.2794
  484.8653
  539.9464
  607.1211
  634.2153
  705.5113
  744.9982];

T1d = [1.17467
    7.03221
    14.8740
    26.7737
    41.6816
    60.5432
    81.3575
```

```
    108.3042
    133.7348
    167.9602
    202.6555
    243.7716
    284.7327
    329.2001
    378.9953
    442.3498
    501.3712
    597.7224
    653.3349
    700.3451
    785.6923
    872.3309
    981.1001
    1023.6778
    1143.3077
    1210.7644];

%codefoto 2
T2 = [0.0831
    0.2229
    0.4820
    0.8517
    1.3407
    1.9282
    2.5956
    3.4195
    4.3111
    5.3877
    6.4564
    7.6308
    8.9189
   10.4991
   11.8998
   13.7632
   15.2430
   17.1376
   19.1314
   21.1885
   23.7241
   29.7873
   31.7547
   34.4910
   35.2436
   36.4763];

T2d = [0.0872
    0.2491
    0.5064
    0.9060
    1.4043
    2.0704
    2.7086
    3.5531
    4.5424
    5.5168
    6.7633
```

```
    7.9836
    9.4360
   10.9265
   12.5478
   14.4560
   16.2536
   18.5528
   20.5429
   22.5710
   25.0178
   28.1791
   30.2885
   33.0819
   35.9263
   39.8586];

%codefoto 3
T3 = [0.1507
    0.2432
    0.5229
    0.9373
    1.5195
    2.2608
    3.2100
    4.4938
    5.9327
    7.4981
    9.3202
   11.5533
   14.2727
   16.7026
   20.1104
   23.5510
   26.9435
   31.4948
   35.6560
   40.4380
   46.3051
   51.7158
   57.4900
   63.8277
   73.4996
   80.0643];

T3d = [0.0620
    0.2208
    0.4884
    0.8688
    1.3790
    1.9641
    2.8647
    3.5823
    4.5021
    5.6187
    6.7090
    8.1194
    9.3831
   11.0526
   12.8849
```

```
    15.0645
    16.3568
    18.9721
    21.4749
    23.9316
    26.4180
    28.2523
    31.8825
    33.1355
    34.7520
    39.4398];

k = 4608/2592;
X = sqrt(k)*[(100:100:2500)';2592];
x = sqrt(X_*Y_);
if metodo==1
    tc = mspline3([X factor*T1],0,0,x);
    td = mspline3([X factor*T1d],0,0,x);
end
if metodo==2
    tc = mspline3([X factor*T2],0,0,x);
    td = mspline3([X factor*T2d],0,0,x);
end
if metodo==3
    tc = mspline3([X factor*T3],0,0,x);
    td = mspline3([X factor*T3d],0,0,x);
end
```

7. Bibliografía

[1] J. Gómez (2010), *Matemáticos, espías y piratas informáticos (Codificación y Criptografía)*, España, RBA Coleccionables S.A.

[2] *Crypt4you. Aula virtual. Introducción a la seguridad informática y criptografía clásica*. http://www.criptored.upm.es/crypt4you/temas/criptografiaclasica/leccion1.html

[3] G. J. Simmons (1992). *A survey of Information Authentication". Contemporary Cryptology, The science of information integrity*. Ed. GJ Simmons, IEEE Press, New York.

[4] G. Zuñiga, F. E. López, R. F. Quenta. (2019) *Criptografía con matrices, el cifrado de Hill*. Criptografía en Algebra Lineal.

[5] Hellman, M. E. (2002), *An overview of public key cryptography*". IEEE Communications Magazine, 40 (5): 42–49.

[6] S. Attaway (2019). *MATLAB: A Practical Introduction to Programming and Problem Solving*. Ed. Butterworth-Heinemann.

[7] Matlab App Designer. https://www.mathworks.com/help/matlab/app-designer.html

[8] E. Bujalance, J. A. Bujalance, A. F. Costa, E. Martínez, (2005). *Elementos de matemática discret*a. Editorial Sanz y Torres, 3a Edición.

[9] Hillier, F.S. y Lieberman, G.J., (2003), *Introducción a la Investigación de Operaciones*, 5ª. Edición, McGrawHill/Interamericana de México, S.A. de C.V., México.

[10] H. Peitgen, H. Jürgens y D. Saupe. (2004). *Chaos and fractals. News frontiers of the science*. Ed. Springer Verlag, Hardcover.

[11] Prof. Herbert Hoeger. *Simulación: Probando generadores de números aleatorios*. http://webdelprofesor.ula.ve/ingenieria/hhoeger/

[12] https://labmatlab2.upv.es/webapps/home/code.html

[13] https://labmatlab2.upv.es/webapps/home/decode.html

Parte 2

PRESUPUESTO

8. Presupuesto del proyecto

A continuación, estudiaremos el presupuesto del proyecto. Una parte fundamental del TFG es valorar la viabilidad económica del proyecto, y aunque se trata de un trabajo académico más que industrial, sigue siendo un proyecto de ingeniería, con unos costes asociados, que a continuación expondremos en este presupuesto.

Este presupuesto se realizará tomando en consideración las "Recomendaciones en la Elaboración de Presupuestos en Actividades de I+D+I" (revisión de 2018) de la UPV de acuerdo con el artículo 83 de la Ley Orgánica de Universidades.

8.1 Cuadro de precios básicos

A continuación, trataremos los precios básicos de los recursos que intervienen en el proyecto. Hemos definido que el autor del TFG, considerado Graduado en Tecnologías Industriales, con un coste de 20€/h. Por otro lado, el tutor del TFG, considerado un Ingeniero Industrial, cuesta 30€/h. El ordenador utilizado principalmente en proyecto es un portátil Lenovo modelo Z50-70, con un precio de 799€, con un periodo de amortización de 6 años, por tanto, considerando que en 2020 tiene 253 días laborables, y la jornada laboral tiene 8 horas, definiremos su coste relativo a la hora de uso de la siguiente forma:

$$Coste\ (€) = \frac{coste\ del\ concepto\ (€)}{periodo\ de\ amortización\ (años) * 253 * 8}$$

De esta misma forma, y considerando un periodo de amortización de 6 años, para el resto de los recursos al tratarse de equipos y aplicaciones informáticas, siendo estos: otro ordenador portátil de la marca MSI modelo PE72 8RD y las licencias de Microsoft, MATLAB y Microsoft Office. Obtenemos de esta forma la siguiente tabla de precios por cada hora de uso de la maquinaria y software.

Concepto	Periodo de amortización (años)	Coste total (€)	Coste (€/h)
Ordenador portátil Lenovo Z50-70	6	799,00	0,0658
Ordenador portátil MSI PE72 8RD	6	849,00	0,0699
Licencia de Windows 10	6	62,00	0,0051
Licencia de MATLAB	6	2.000,00	0,1647
Licencia de Microsoft Office	6	69,00	0,0057

Tabla 2. Costes por hora de maquinaria y software

Código	Unidad	Descripción	Precio (€/h)
O1	h	Graduado en GITI	20,00
O2	h	Ingeniero Industrial	30,00
H1	h	Ordenador portátil Lenovo Z50-70	0,07
H2	h	Ordenador portátil MSI PE72 8RD	0,07
S1	h	Licencia de Windows 10	0,01
S2	h	Licencia de MATLAB	0,16
S3	h	Licencia de Microsoft Office	0,01
M1	ud	Folios	0,01
M2	ud	Bolígrafos	1,00
M3	ud	Memoria USB	8,00

Tabla 3. Cuadro de precios básicos

8.2 Cuadro de precios unitarios descompuestos

A continuación, tenemos los precios unitarios descompuestos del proyecto, referidos a cada capítulo en el que dividimos la realización del trabajo. Se supone un 2% de gastos directos complementarios, para tener en cuenta posibles gastos auxiliares como el contrato de la luz o la conexión a Internet.

Código	Ud.	Descripción	Precio	Rdto.	Importe (€)
Capítulo 1: Estudio previo					
1.1	h	Estudio teórico del proyecto			
O1	h	Graduado en GITI	20,00	1	20,00
O2	h	Ingeniero industrial	30,00	1	30,00
H1	h	Ordenador portátil Lenovo Z50-70	0,07	1	0,07
H2	h	Ordenador portátil MSI PE72 8RD	0,07	1	0,07
S1	h	Licencia de Windows 10	0,01	2	0,01
	%	Costes directos complementarios	50,15	0,02	1,00
			Coste total		**51,15**

Tabla 4. Unidades de obra del Capítulo 1

Código	Ud.	Descripción	Precio	Rdto.	Importe (€)
Capítulo 2: Realización del proyecto					
2.1	h	Programación del sistema de codifciación			
O1	h	Graduado en GITI	20,00	1	20,00
O2	h	Ingeniero industrial	30,00	1	30,00
H1	h	Ordenador portátil Lenovo Z50-70	0,07	1	0,07
H2	h	Ordenador portátil MSI PE72 8RD	0,07	1	0,07
S1	h	Licencia de Windows 10	0,01	2	0,01
S2	h	Licencia de Matlab	0,16	2	0,33
CDC	h	Costes directos complementarios	50,48	0,02	1,01
			Coste total		**51,48**
2.2	h	Redacción de la memoria			
O1	h	Graduado en GITI	20,00	1	20,00
H1	h	Ordenador portátil Lenovo Z50-70	0,07	1	0,07
S1	h	Licencia de Windows 10	0,01	1	0,01
S3	h	Licencia de Microsoft Office	0,01	1	0,01
CDC	h	Costes directos complementarios	20,08	0,02	0,40
			Coste total		**20,48**
2.3	ud	Gastos en material fungible			
M1	ud	Folios	0,01	10	0,10
M2	ud	Bolígrafos	1,00	3	3,00
M3	ud	Memoria USB	8,00	1	8,00
CDC	ud	Costes directos complementarios	11,10	0,02	0,22
			Coste total		**11,32**

Tabla 5. Unidades de obra del Capítulo 2

8.3 Mediciones y cuadro de presupuestos parciales

Suponiendo una duración aproximada del proyecto de unas 290 horas, estimamos que se han dedicado unas 50 horas al estudio teórico previo, 130 a la programación del sistema de codificación y 110 a la redacción de la memoria.

Código	Ud.	Descripción	Medición	Precio	Importe (€)
Capítulo 1: Estudio previo					
1.1	h	Estudio teórico del proyecto	50	51,15	2.557,44
			Coste total		2.557,44
Capítulo 2: Realización del proyecto					
2.1	h	Programación del sistema de codificación	130	51,48	6.693,02
2.2	h	Redacción de la memoria	110	20,48	2.252,59
2.3	ud	Gastos en material fungible	1	11,32	11,32
			Coste total		8.956,94

Tabla 6. Cuadro de presupuestos parciales

8.4 Resumen del presupuesto

La definición general del presupuesto es la siguiente:

PRESUPUESTOS PARCIALES		
Capítulo 1: Estudio previo		2.557,44 €
Capítulo 2: Realización del proyecto		8.956,94 €
PRESUPUESTO DE EJECUCIÓN MATERIAL (PEM)		**11.514,38 €**
Gastos generales (13%)		1.496,87 €
Beneficio industrial (6%)		690,86 €
PRESUPUESTO DE EJECUCIÓN POR CONTRATA (PEC)		**13.702,11 €**
IVA (21%)		2.877,44 €
PRESUPUESTO BASE DE LICITACIÓN (PBL)		**16.579,56 €**

Tabla 7. Presupuesto General

El presupuesto total asciende a:

DIECISEISMIL QUINIENTOS SETENTA Y NUEVE EUROS CON CINCUENTA Y SEIS CÉNTIMOS