

# **Chaotic Dynamics Analysis and its application in Cryptography**



**Cluster Innovation Centre  
University of Delhi**

July 2023

Month Long Project submitted for  
**Engineering Through Linear Algebra**

## **Acknowledgment**

We would like to express our special thanks of gratitude to our mentor **Dr Sonam Tanwar** for her guidance and supervision as well as providing necessary information regarding the project. It would be a great pleasure for me to present our month long project on **Engineering Through Linear Algebra**.

## **Certificate of Originality**

The work embodied in this report entitled “**Chaotic Dynamics Analysis and its application in Cryptography**” has been carried out by Saswat Susmoy, Aryan Sharma and Tushitaa Narayan Ojha for the paper “Engineering Through Linear Algebra”. We declare that the work and language included in this project report are free from any kind of plagiarism.

**Dr Sonam Tanwar**

Assistant Professor  
Department of Mathematics  
Cluster Innovation Centre  
University of Delhi

## **Certificate of Completion**

This is to certify that the following persons: **Saswat Susmoy Sahoo, Aryan Sharma and Tushitaa Narayan Ojha** have completed this Month Long Project for "Engineering Through Linear Algebra" under my guidance and supervision as per the contentment of the requirements of the second semester in the course B.Tech (Information Technology and Mathematical Innovations) at the Cluster Innovation Centre, University of Delhi.

**Dr Sonam Tanwar**

Assistant Professor  
Department of Mathematics  
Cluster Innovation Centre  
University of Delhi

# **Index**

<b>01</b>	Abstract	<b>00</b>
<b>02</b>	Introduction	<b>01</b>
<b>03</b>	Methodology	<b>03</b>
<b>04</b>	Creations of PRNG	<b>07</b>
<b>05</b>	Ciphers Used	<b>08</b>
<b>06</b>	Practical Applications	<b>14</b>
<b>07</b>	References	<b>15</b>

## **Abstract**

# **Chaotic Dynamics Analysis and its application in Cryptography**

by

**Saswat Susmoy Sahoo || Aryan Sharma || Tushitaa Narayan Ojha**

Cluster Innovation Centre, 2023

In this paper, We are interested in the contribution of Linear Algebra in the analysis of chaotic behaviour of the Logistic map. Though there is no direct way to apply linear algebra for chaos analysis, but an approximation can help in knowing whether a system is chaotic or not. This can be further checked with non-linear techniques to say surely that system is chaotic. Once the analysis is done, chaotic systems can be used in developing efficient PRNGs (Pseudo Random Number Generators) which have significant role in cryptography. An efficient and reliable PRNG can boost the security of any cryptographic algorithm. That's why we developed our own PRNG function in MATLAB using our findings in the analysis of Chaotic dynamics of the logistic map.

# **Introduction**

## ***1.1 Background and Context***

Chaotic dynamics is a fascinating field of study that explores the behavior of complex systems that are highly sensitive to initial conditions. The logistic map is a simple mathematical model widely used to investigate chaotic behavior in dynamical systems. It describes the population growth or decline of a species in discrete time, considering factors such as reproduction and limited resources. Eigenvalue analysis, on the other hand, focuses on the study of the eigenvalues and eigenvectors of a matrix, providing valuable insights into the stability and behavior of linear systems.

## ***1.2 Scope and Objectives***

The scope of the project "Chaotic Dynamics and Eigenvalue Analysis in the Logistic Map" is to explore and analyze the chaotic behavior exhibited by the logistic map and its relationship with eigenvalue analysis. The objectives of the project are as follows:

1. Implement the logistic map equation in MATLAB to simulate the population dynamics over time.
2. Perform numerical analysis to calculate the Lyapunov exponents, which quantify the rate of divergence of nearby trajectories, indicating chaotic behavior.
3. Conduct bifurcation analysis to observe the emergence of periodic orbits and period-doubling bifurcations as the parameter varies.
4. Utilize eigenvalue analysis techniques to investigate the stability and behavior of the logistic map using single value decomposition.
5. Explore the applications of chaotic systems in PRNGs and Cryptography.
6. Visualize and interpret the results, generating graphs and diagrams to illustrate the chaotic dynamics and its applications.

### ***I.3 Achievements***

We have successfully created a simulation to determine the state of a function and used it to produce random generated numbers using PRNG (Pseudo Random Number Generator).

### ***II.1 Problem Statement***

The problem addressed in this paper is the analysis of chaotic behavior in the logistic map and the utilization of linear algebra to approximate and confirm chaos presence. While direct application of linear algebra for chaos analysis is not feasible, an approximation technique is employed to determine whether a system is chaotic. The paper aims to investigate the contribution of linear algebra in chaos analysis, validate the approximation technique using non-linear methods, and subsequently develop a customized Pseudo Random Number Generator (PRNG) based on the chaotic dynamics of the logistic map. The goal is to enhance the security of cryptographic algorithms through the utilization of efficient and reliable PRNGs generated from chaotic systems.

### ***Pre-Requisites***

Theoretical

- Linear Algebra
- Determinants and Matrices
- MATLAB Programming & Data Visualization



## **II.2 Methodology**

## ***Chaos dynamics of the Logistic map***

### ***Understanding Logistic map***

The logistic map is famous for two reasons: it gives a way of predicting how a population of animals will grow or shrink over time, and it illustrates the fascinating phenomenon of mathematical chaos.

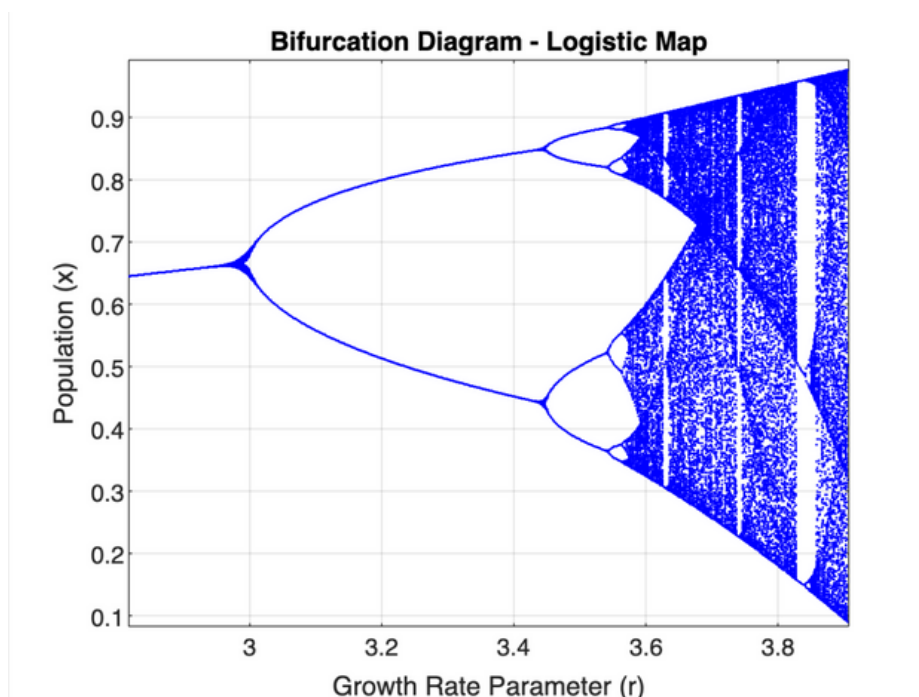
It is represented by a simple equation :-

$$X_{n+1} = r * X_n * (1 - X_n)$$

Here,  $r$  is growth parameter, and  $0 \leq X \leq 1$ .

### ***Behaviour of logistic map***

Lets say we chose  $r=2$  and initial value of  $x$  as  $0.5$ . After doing some iterations we will see that on further iterations  $x$  is not growing but stable at a constant value. This repeats for  $r$  ranging from  $1$  to  $3$ . What if  $r$  is  $>3$ , then after some iterations  $x$  is oscillating its value between two values. As we go on to increase  $x$  is oscillating between  $4$ ,  $8$ ,  $16$  and after some time its total chaos. This is what represented in the bifurcation diagram for different values of  $r$  and the trajectory of  $x$ .

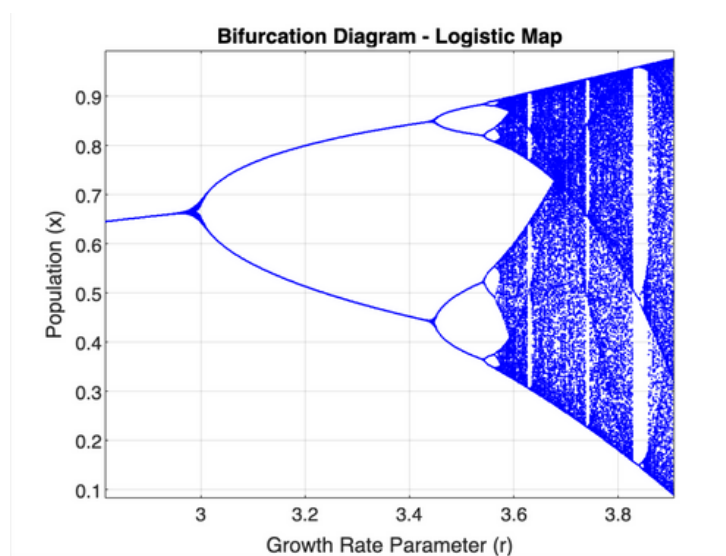


## *Bifurcation Diagram - Logistic Map Code*

```

1 % Bifurcation Diagram - Logistic Map
2 clc
3 clear all
4 % Parameters
5 numIterations = 200; % Number of iterations
6 r_values = linspace(2.8, 4.5, 1000); % Range of growth rate parameter
7 x0 = 0.5; % Initial condition
8 % Initialize vectors
9 x = zeros(numIterations, 1); % Vector to store population values
10 % Iterate over different values of r
11 for i = 1:length(r_values)
12     r = r_values(i); % Current growth rate parameter
13     x0=0.5;
14     for j=1:100
15         x0 = r*x0*(1-x0);
16     end
17     x(1) = x0; % Reset initial condition
18
19     % Iterate logistic map equation
20     for n = 1:numIterations-1
21         x(n+1) = r * x(n) * (1 - x(n));
22     end
23     % Plotting
24     plot(r_values(i) * ones(numIterations, 1), x, '.', 'Color', 'b', 'MarkerSize', 1);
25     hold on;
26 end
27 xlabel('Growth Rate Parameter (r)');
28 ylabel('Population (x)');
29 title('Bifurcation Diagram - Logistic Map');
30 grid on;

```



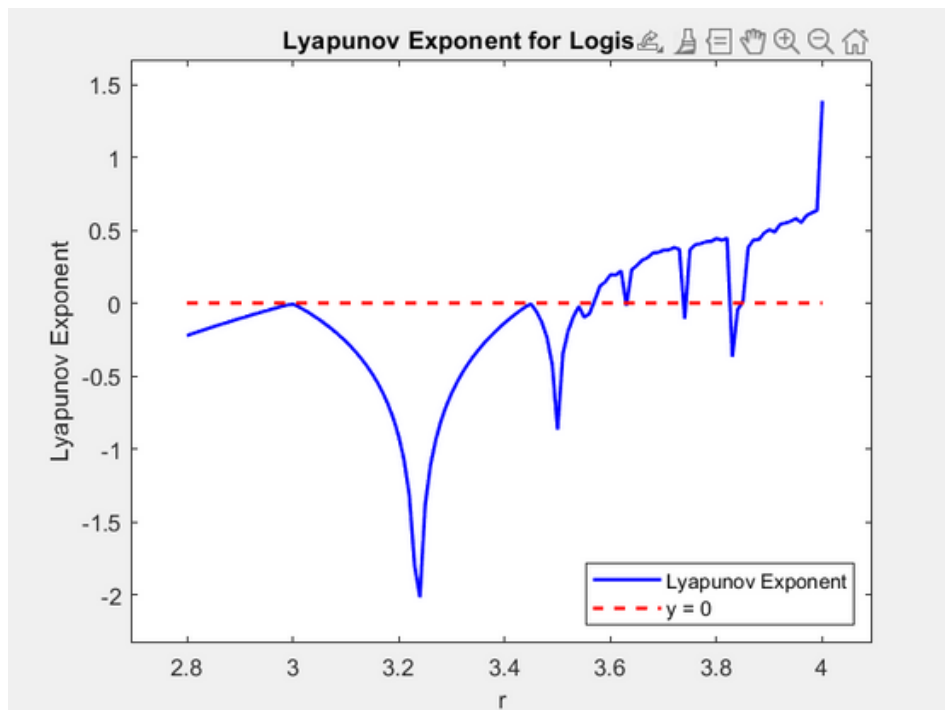
Graph generated by the code

## Chaotic dynamics analysis

So how do we find where the system is chaotic or where it is not. There is a way to do this using Lyapunov exponent which is numerically calculated as

$$\lambda_L = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=1}^n \ln \left| f'(x_i) \right|$$

where  $f(x)$  is the logistic map equation.



This figure shows that the the system is chaotic after 3.5 but at some instants become non-chaotic. For safe side we can choose  $r > 4$  for chaotic behaviour.

## *Chaotic dynamics analysis (Linear Algebra Technique)*

Another way to calculate lyapunov exponent using linear algebra techniques is by Finding the jacobian matrix (partial derivatives of logistic map at a particular  $r$ ). Using single value Decomposition to find the sigma matrix. The largest eigenvalue is obtained with help of this singular matrix which represents the approximate lyapunov exponent at that instant.

If lyapunov exponent is positive the system is chaotic, if it is 0 at some point it means bifurcation is happening at that instant. If it is negative the system is not chaotic

```

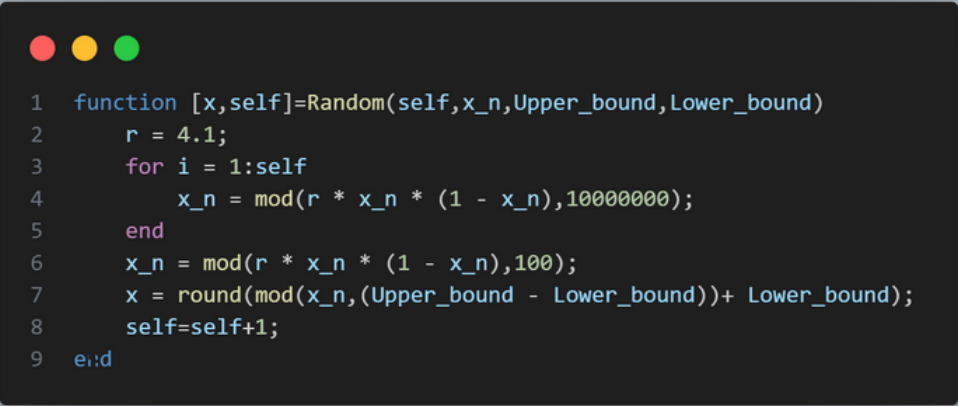
1  % Logistic map parameters
2  r = 4; % Parameter r
3  nIterations = 1000; % Number of iterations
4  % Initialize variables
5  x = zeros(1, nIterations);
6  x(1) = 0.5; % Initial condition
7  % Compute the time series
8  for i = 2:nIterations
9      x(i) = r * x(i-1) * (1 - x(i-1));
10 end
11 % Compute the Jacobian matrix
12 J = zeros(nIterations-1, 1);
13 for i = 1:(nIterations-1)
14     J(i) = r - 2*r*x(i);
15 end
16 % Compute singular value decomposition
17 [U, S, V] = svd(J);
18 % The largest Lyapunov exponent is given by the logarithm of the largest singular value
19 lambda = log(S(1, 1))/r;
20 % Display the estimated Lyapunov exponent
21 disp(['Estimated Lyapunov exponent: ', num2str(lambda)]);

```

Lyapunov Linear Code

## *Application of this analysis*

After we know that when the system is chaotic which is for  $r > 4$  we have developed a PRNG function which takes  $x_n$  as input value and generates random value on the trajectory odd  $x_n$  for  $r = 4.1$ . For every call this function updates a variable named `self` which is used to move forward in trajectory after every function call.



```

1 function [x,self]=Random(self,x_n,Upper_bound,Lower_bound)
2     r = 4.1;
3     for i = 1:self
4         x_n = mod(r * x_n * (1 - x_n),10000000);
5     end
6     x_n = mod(r * x_n * (1 - x_n),100);
7     x = round(mod(x_n,(Upper_bound - Lower_bound))+ Lower_bound);
8     self=self+1;
9 end

```

Random PRNG Generator

## *Using this PRNG in our encryption model*

We have developed a simple cryptographic model which uses basic block ciphers and apply them randomly in any order while encrypting and decrypts using the reverse order. Each of the algorithm has been improved while implementing individually. We have only used our developed PRNG function wherever required in individual ciphers and main encryption file.

## *Ciphers used*

- Vigenere cipher
- Shift cipher
- Substitution cipher

### *Shift cipher*

**Shift cipher** is a monoalphabetic substitution cipher. It's also sometimes referred to as Caesar Shift as it was used by Julio Caesar himself.

**Encryption:** When a plaintext is provided to this algorithm, we use the pre-defined ASCII values to represent every character. A random key is generated by the system from the ASCII table. To generate the encrypted text the original plain text after being encoded with the ASCII counterparts is shifted using an algorithm where the random key is added to the encoded plaintext and then operated with the modulus function as shown below:

$$E(x) = [f(x) + k] \bmod 127$$

where,  $E(x)$  is encrypted text

$f(x)$  is the encoded plaintext

$k$  is the generated random key

**Decrypting:** When a ciphertext is provided to this algorithm, we use the previously generated key (the one used to encrypt the plaintext) and reverse the encryption process. We start by backward shifting (subtracting) the ciphertext with the key and then applying the modulus function with 127 as shown below:

$$D(x) = [E(x) - k] \bmod 127$$

where,  $D(x)$  is decrypted plain text

**Note:** As the ASCII Table from 1 to 32 contains characters like space, enter etc, we have made an attempt to salt the code by adding 33 to the plaintext and then applied the Encryption technique.

```

1 function [string,self] = ShiftEncryption(self,x_n,message)
2     [x,self]=Random(self,x_n,0,1000);
3     fileID=fopen('keys/shiftcipherKEY.txt','w');
4     fprintf(fileID,'%d',x);
5     fclose(fileID);
6     prob=[];
7     ciphered=[];
8     plain=double(message);
9     for i=1:length(plain)
10         ciphered(i)=mod(plain(i)+x,127);
11         if ciphered(i)<33
12             ciphered(i)=ciphered(i)+33;
13             prob=[prob, i];
14         end
15     end
16
17     writematrix(prob,'keys/shiftcipherprob.txt')
18     string=char(ciphered);
19     self=self;
20
21 end

```

## Shift Cipher Encryption

```

1 function [string,self] = ShiftEncryption(self,x_n,message)
2     [x,self]=Random(self,x_n,0,1000);
3     fileID=fopen('keys/shiftcipherKEY.txt','w');
4     fprintf(fileID,'%d',x);
5     fclose(fileID);
6     prob=[];
7     ciphered=[];
8     plain=double(message);
9     for i=1:length(plain)
10         ciphered(i)=mod(plain(i)+x,127);
11         if ciphered(i)<33
12             ciphered(i)=ciphered(i)+33;
13             prob=[prob, i];
14         end
15     end
16
17     writematrix(prob,'keys/shiftcipherprob.txt')
18     string=char(ciphered);
19     self=self;
20
21 end

```

## Shift Cipher Decryption



## **Vigenere cipher**

**Vignere Cipher:** What is today known as the Vigenère Cipher was actually first described by Giovan Battista Bellaso in his 1553 book *La cifra del. Sig. Giovan Battista Bellaso*

**Encryption:** In Vignere Cipher we generate a list of random numbers and the length of list is also random. This list of numbers is called the random number list. And every alphabet of our text is represented by its decimal value according to ASCII. And thus we get our message list. Then we simply add the random number list and the message list. To get a encrypted list containing numbers and then each number is converted to its character value according to ASCII. In this way we get our encrypted characters.

**Decryption:** In order to decrypt the encrypted matrix, we first convert the encrypted characters list into encrypted numbers list. Then we subtract the random number list from this encrypted number list to get our message list containing numbers which is then converted into characters to get our final message text.

```

1 function [string,self]=vignereEncryption(self,x_n,message)
2     [keyListSize,self]=Random(self,x_n,length(message),3);
3     keylist=[];
4     for i=1:keyListSize
5         [keylist(i),self]=Random(self,x_n,127,33);
6     end
7     plain=double(message);
8     for j=1:(length(message)-length(keylist))
9         keylist=[keylist,keylist(j)];
10    end
11    writematrix(keylist,'keys/vignerekeylist.txt');
12    prob=[];
13    encryptedList=[];
14    for k=1:length(message)
15        y=mod(plain(k)+keylist(k),127);
16        if y<33
17            y=y+33;
18            prob=[prob k];
19        end
20        encryptedList(k)=y;
21    end
22    writematrix(prob,'keys/vignereprob.txt');
23    string=char(encryptedList);
24    self=self;
25 end

```

## Vigenere Cipher Encryption

```

1 function string= vignereDecryption(message)
2     keylist=readmatrix('keys/vignerekeylist.txt');
3     prob=readmatrix('keys/vignereprob.txt');
4     y=double(message);
5     plain=[];
6     for i=1:length(keylist)
7         for j=1:length(prob)
8             if i==prob(j)
9                 y(i)=y(i)-33;
10            end
11        end
12        plain(i)=mod(y(i)-keylist(i),127);
13    end
14    string=char(plain);
15 end

```

## Vigenere Cipher Decryption

## **Substitution cipher**

### **Substitution Cipher:**

**Encryption:** In this algorithm we have two lists.

List 1 which stores the integer values from 32 to 126.

And List 2 of length equal to list 1 which stores random integers from 32 to 127 .

When a plain text is provided to this algorithm, we use pre-defined ASCII values to represent every character. We check the index from list 1 at which this ASCII value is present.

Then we checks what value is present at that index in list 2 and stores them in a new list. We get our encrypted text by using the character values of the ASCII values in this new list.

**Decryption:** In order to decrypt the encrypted text , we use previously generated list 2 and the list 1. We first convert the characters to its corresponding ASCII values. Then we check the index of each of these values in List 2 and checks the values at these indexes in list 1.

We then convert those ASCII values to get Decrypted characters.

```

1 function [string,self] = substitutuionencryption(self,x_n,message)
2     list1 = uint32(32):uint32(126);
3     list2 = [];
4     for i=1:10000000
5         [r,self]=Random(self,x_n,126,32);
6         for j=1:length(list2)
7             if r==list2(j)
8                 break
9             end
10        end
11        if r==list2(j)
12            continue
13        end
14        list2=[list2 ,r];
15        if length(list2)==length(list1)
16            break
17        end
18    end
19
20    writematrix(list1,'keys/subsitutionlist1.txt');
21    writematrix(list2,'keys/subsitutionlist2.txt');
22    plain=double(message);
23
24    indexinlist1=[];
25    for k=1:length(plain)
26        for l=1:length(list1)
27            if plain(k)==list1(l)
28                indexinlist1=[indexinlist1,l];
29                break
30            end
31        end
32    end
33    cipherindex=[];
34    for m=1:length(plain)
35        cipherindex(m)=list2(indexinlist1(m));
36    end
37    string=char(cipherindex);
38    self=self;
39 end

```

## Substitution Cipher Encryption

```

1 function [string,self] = ShiftEncryption(self,x_n,message)
2     [x,self]=Random(self,x_n,0,1000);
3     fileID=fopen('keys/shiftcipherKEY.txt','w');
4     fprintf(fileID,'%d',x);
5     fclose(fileID);
6     prob=[];
7     ciphered=[];
8     plain=double(message);
9     for i=1:length(plain)
10        ciphered(i)=mod(plain(i)+x,127);
11        if ciphered(i)<33
12            ciphered(i)=ciphered(i)+33;
13            prob=[prob, i];
14        end
15    end
16
17
18    writematrix(prob,'keys/shiftcipherprob.txt')
19    string=char(ciphered);
20    self=self;
21 end

```

## Substution Cipher Decryption

## ***Other Practical Applications***

**1. Cryptography:** Chaotic systems, such as the logistic map, can be used to generate secure encryption keys. The project can be extended to develop a secure encryption algorithm based on the chaotic dynamics of the logistic map. By analyzing the eigenvalue spectrum and chaotic behavior, you can design a robust encryption scheme for secure data transmission.

**2. Random Number Generation:** Chaotic systems exhibit complex and unpredictable behavior, making them suitable for generating random numbers. The logistic map project can be utilized to develop a random number generator that can find applications in simulations, Monte Carlo methods, cryptography, and scientific computing.

**3. Nonlinear Dynamics:** Studying the chaotic behavior of the logistic map helps in understanding nonlinear dynamic systems. The project can serve as a basis for exploring and analyzing the properties of chaotic systems, which have applications in physics, biology, economics, and other disciplines that involve complex phenomena.

**4. Signal Processing:** Chaotic systems can be employed in various signal processing applications. By understanding the chaotic dynamics of the logistic map and performing eigenvalue analysis, you can explore applications such as chaos-based communication systems, secure data transmission, and noise reduction techniques.

**5. Image Encryption:** Chaotic systems can be used for image encryption to provide robust protection against unauthorized access. By applying the principles of chaotic dynamics and eigenvalue analysis to image encryption algorithms, you can develop secure and efficient image encryption techniques.

## **References**

- Lyapunov Functions and Stability in Control Theory by Lionel Rossier & Andrea Bacciotti
- Pseudorandom Number Generator (PRNG) Design Using Hyper-Chaotic Modified Robust Logistic Map (HC-MRLM) by Muhammad Irfan, Asim Ali & Waqar Ahmad
- A Review Paper on Cryptography by Abdalbasit Mohammed & Nuryahat Varol