NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DOUBLE PENDULUM CHAOTIC MODEL FOR PSEUDORANDOM NUMBER GENERATION

by

Ryan C. Hard

March 2020

Thesis Advisor: Thor Martinsen
Second Reader: Beny Neta

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE DOUBLE PENDULUM CHAOTIC MODEL FOR PSEUDORANDOM NUMBER GENERATION	**5. FUNDING NUMBERS**
6. AUTHOR(S) Ryan C. Hard	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

The double pendulum is a system of two connected masses, one tethered to a point in space and the other tethered to the first mass. The double pendulum exhibits chaotic motion under the influence of an external force such as the gravitational force. The chaotic motion is sensitive to the initial conditions or positions of the masses, resulting in an infinite number of possible motion paths. The chaotic motion paths produced by the double pendulum can be exploited to produce binary sequences. This thesis focuses on determining if the chaotic motion of the double pendulum can be used as a pseudorandom number generator (PRNG) by modeling the motion and using different methods of extracting bits from the motion paths to produce pseudorandom binary sequences. The pseudorandom binary sequences are then evaluated using tests for randomness as defined by the National Institute of Standards and Technology (NIST). The methods of bit extraction can then be compared based on their NIST test results to ultimately determine the practicality of the double pendulum as a PRNG. Considerations can be made for different methods of bit extraction or the use of different chaotic motions.

14. SUBJECT TERMS pseudorandom number generation, chaotic motion, double pendulum	**15. NUMBER OF PAGES** 67
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

DOUBLE PENDULUM CHAOTIC MODEL FOR PSEUDORANDOM NUMBER GENERATION

Ryan C. Hard
Lieutenant, United States Navy
BS, Rutgers University, 2012
BA, Rutgers University, 2012

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
March 2020

Approved by: Thor Martinsen
Advisor

Beny Neta
Second Reader

Wei Kang
Chair, Department of Applied Mathematics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The double pendulum is a system of two connected masses, one tethered to a point in space and the other tethered to the first mass. The double pendulum exhibits chaotic motion under the influence of an external force such as the gravitational force. The chaotic motion is sensitive to the initial conditions or positions of the masses, resulting in an infinite number of possible motion paths. The chaotic motion paths produced by the double pendulum can be exploited to produce binary sequences. This thesis focuses on determining if the chaotic motion of the double pendulum can be used as a pseudorandom number generator (PRNG) by modeling the motion and using different methods of extracting bits from the motion paths to produce pseudorandom binary sequences. The pseudorandom binary sequences are then evaluated using tests for randomness as defined by the National Institute of Standards and Technology (NIST). The methods of bit extraction can then be compared based on their NIST test results to ultimately determine the practicality of the double pendulum as a PRNG. Considerations can be made for different methods of bit extraction or the use of different chaotic motions.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

List of Acronyms and Abbreviations

NIST National Institute of Standards and Technology

PRNG Pseudorandom Number Generator

LFSR Linear Feedback Shift Register

DOD Department of Defense

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I would like to thank my thesis advisor, Commander Thor Martinsen, for his guidance throughout the development of this thesis as well as for giving me the opportunity to work on such an interesting topic.

I would like to thank my second reader, Professor Beny Neta, for his mathematical proof-reading and guidance on the development of this thesis.

I would like to thank my friend and colleague, Lieutenant Alex Gutzler, for his assistance in fixing the Python codes I would inevitably break so that the results of this thesis could exist.

I would also like to thank my wife for her love and support, as well as for enduring the times I would lock myself away to work on this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

1.1 Background

The Department of Defense (DOD) is reliant on secure communications and the pervasive use of pseudorandom binary sequences in modern communications technology. Cryptographic security depends on the use and generation of pseudorandom numbers. A pseudorandom number generator (PRNG) creates a pseudorandom number based on a seed and is then used in the encryption process as the key. In order for the encryption to be secure, the PRNG must be unpredictable so as to prevent an attacker from deriving what the key could be based on observed bits. Chaotic systems exhibit unpredictable behavior that could be exploited for use in pseudorandom number generation. As stated in [1], chaotic systems are extremely sensitive to initial conditions, meaning that for a marginal change from one initial condition to another, the resulting behaviors based on each set of initial conditions will differ drastically between the two. Chaotic systems may be suitable for pseudorandom number generation given their unpredictability and sensitivity to initial conditions.

Various chaotic systems are described in [2], such as Duffing oscillators, Van der Pol oscillators, and double pendulums. Of these systems, the double pendulum is comparatively simple to model. This thesis focuses on using the chaotic motion of the double pendulum to develop a PRNG. The second chapter of this thesis describes the double pendulum and derives the equations of motion to be used in a model. The third chapter covers the methodology by which we attempt to create pseudorandom sequences from the chaotic motion of the double pendulum and then test the sequences for randomness using a statistical test suite designed by the National Institute of Standards and Technology (NIST). The fourth chapter describes the results of the NIST tests, along with implications associated with the various testing scenarios. The fifth chapter discusses the conclusion of our testing as well as possible future work. All coding is conducted using MATLAB and Python, where the associated codes can be found in the appendices.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
The Double Pendulum

In this chapter, we describe and derive the equations of motion for the double pendulum. The derivation is based on Appendix B of [1].

2.1 Description

In two-dimensional space, the double pendulum consists of two masses, m_1 and m_2. m_1 is attached to a fixed point in space by the first arm with a specified length, l_1. The first mass is attached to the second mass by a second arm with a specified length, l_2. For simplicity, we assume that the arms do not have mass; that gravity, $g = 9.8\ m/s^2$, only acts on the two masses; that no friction exists anywhere in the system. We assume the system exists in the xy-plane of the Cartesian coordinate system, and that gravity acts in the negative y-direction. We choose the origin as the fixed point to which the first arm is attached. The angle θ_1 is measured counterclockwise from the y-axis to the first arm and the angle θ_2 is measured counterclockwise from the vertical line drawn from m_1 to the second arm. See Figure 2.1. The position of m_1 is defined as (x_1, y_1) and the position of m_2 is defined as (x_2, y_2), where

$$x_1 = l_1 sin\theta_1\,,\ y_1 = -l_1 cos\theta_1 \tag{2.1}$$

$$x_2 = l_1 sin\theta_1 + l_2 sin\theta_2\,,\ y_2 = -l_1 cos\theta_1 - l_2 cos\theta_2. \tag{2.2}$$

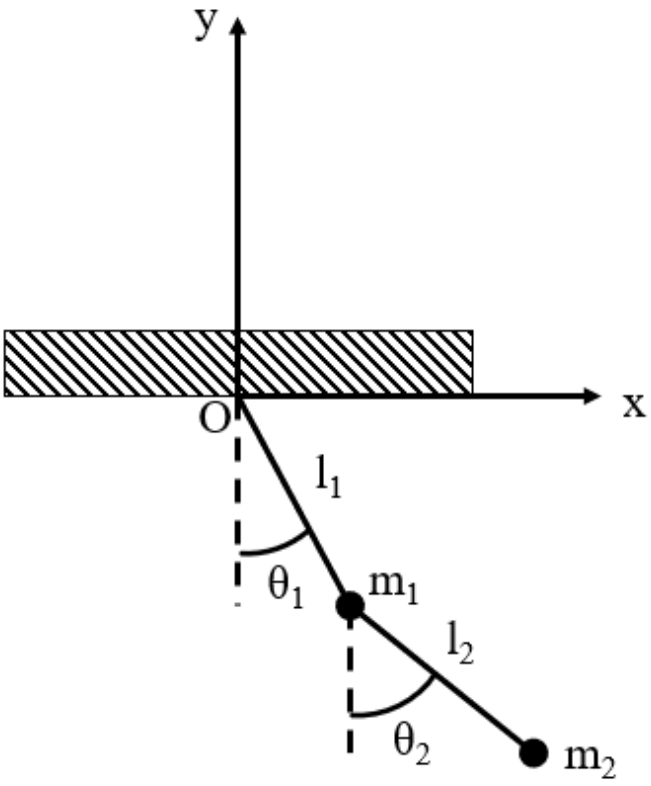

Figure 2.1. The Double Pendulum

2.2 Derivation of Equations

In order to utilize the chaotic motion of the double pendulum, we require the equations of motion for the masses. We derive the equations of motion by finding the Lagrangian of the system and then using the Euler-Lagrange equation to find equations for θ_1 and θ_2, which can then be used in the position equations defined in the previous section. The Lagrangian is defined as

$$L = K - U,$$

where K is the kinetic energy of the system and U is the potential energy of the system. The kinetic energy is

$$K = \frac{1}{2}m_1 v_1^2 + \frac{1}{2}m_2 v_2^2 = \frac{1}{2}m_1(\dot{x}_1^2 + \dot{y}_1^2) + \frac{1}{2}m_2(\dot{x}_2^2 + \dot{y}_2^2). \tag{2.3}$$

From equations (2.1) and (2.2) we find

$$\dot{x}_1 = l_1\dot{\theta}_1 cos\theta_1\,,\ \dot{y}_1 = l_1\dot{\theta}_1 sin\theta_1$$

$$\dot{x}_2 = l_1\dot{\theta}_1 cos\theta_1 + l_2\dot{\theta}_2 cos\theta_2\,,\ \dot{y}_2 = l_1\dot{\theta}_1 sin\theta_1 + l_2\dot{\theta}_2 sin\theta_2.$$

Equation (2.3) then becomes

$$K = \frac{1}{2}m_1 l_1^2\dot{\theta}_1^2 + \frac{1}{2}m_2[l_1^2\dot{\theta}_1^2 + l_2^2\dot{\theta}_2^2 + 2l_1 l_2\dot{\theta}_1\dot{\theta}_2 cos(\theta_1 - \theta_2)].$$

The potential energy is

$$U = m_1 g y_1 + m_2 g y_2 = -m_1 g l_1 cos\theta_1 - m_2 g[l_1 cos\theta_1 + l_2 cos\theta_2].$$

Finally,

$$\begin{aligned} L &= K - U \\ &= \frac{1}{2}m_1 l_1^2\dot{\theta}_1^2 + \frac{1}{2}m_2[l_1^2\dot{\theta}_1^2 + l_2^2\dot{\theta}_2^2 + 2l_1 l_2\dot{\theta}_1\dot{\theta}_2 cos(\theta_1 - \theta_2)] \\ &+ m_1 g l_1 cos\theta_1 + m_2 g[l_1 cos\theta_1 + l_2 cos\theta_2]. \end{aligned}$$

From the Lagrangian, we utilize the Euler-Lagrange equations to find the equations of motion in terms of angular acceleration, which will allow us to develop a set of first-order differential equations to define the model. The Euler-Lagrange equations are

$$\frac{d}{dt}\left(\frac{dL}{d\dot{q}_j}\right) = \frac{dL}{dq_j}, \tag{2.4}$$

where q is the position variable, in this case θ_1 and θ_2. Starting with θ_1, we have

$$\frac{dL}{d\theta_1} = -m_2 l_1 l_2\dot{\theta}_1\dot{\theta}_2 sin(\theta_1 - \theta_2) - (m_1 + m_2)g l_1 sin\theta_1, \tag{2.5}$$

$$\frac{dL}{d\dot{\theta}_1} = m_1 l_1^2\dot{\theta}_1 + m_2[l_1^2\dot{\theta}_1 + l_1 l_2\dot{\theta}_2 cos(\theta_1 - \theta_2)], \text{ and}$$

$$\frac{d}{dt}\left(\frac{dL}{d\dot{\theta}_1}\right) = m_1 l_1^2\ddot{\theta}_1 + m_2[l_1^2\ddot{\theta}_1 + l_1 l_2\ddot{\theta}_2 cos(\theta_1 - \theta_2) - l_1 l_2\dot{\theta}_2 sin(\theta_1 - \theta_2)(\dot{\theta}_1 - \dot{\theta}_2)]. \tag{2.6}$$

Using equations (2.4), (2.5), and (2.6), we solve for $\ddot{\theta_1}$ to obtain

$$\ddot{\theta_1} = \frac{-m_2 l_2 \ddot{\theta_2} cos(\theta_1 - \theta_2) - (m_1 + m_2) g sin\theta_1 - m_2 l_2 \dot{\theta_2^2} sin(\theta_1 - \theta_2)}{(m_1 + m_2) l_1}. \tag{2.7}$$

Similarly, for θ_2 we have

$$\frac{dL}{d\theta_2} = m_2 l_2 [l_1 \dot{\theta_1} \dot{\theta_2} sin(\theta_1 - \theta_2) - g sin\theta_2], \tag{2.8}$$

$$\frac{dL}{d\dot{\theta_2}} = m_2 l_2^2 \dot{\theta_2} + m_2 l_1 l_2 \dot{\theta_1} cos(\theta_1 - \theta_2), \text{ and}$$

$$\frac{d}{dt}\left(\frac{dL}{d\dot{\theta_2}}\right) = m_2 l_2^2 \ddot{\theta_2} + m_2 l_1 l_2 [\ddot{\theta_1} cos(\theta_1 - \theta_2) - \dot{\theta_1} sin(\theta_1 - \theta_2)(\dot{\theta_1} - \dot{\theta_2})]. \tag{2.9}$$

Using equations (2.4), (2.8), and (2.9), we solve for $\ddot{\theta_2}$ to obtain

$$\ddot{\theta_2} = \frac{-g sin\theta_2 - \ddot{\theta_1} l_1 cos(\theta_1 - \theta_2) + \dot{\theta_1^2} l_1 sin(\theta_1 - \theta_2)}{l_2}. \tag{2.10}$$

We then substitute equation (2.7) into equation (2.10) and vice versa to find $\ddot{\theta_1}$ and $\ddot{\theta_2}$ in terms of at most once differentiated variables. This yields

$$\ddot{\theta_1} = \frac{-g sin\theta_1 + M g sin\theta_2 cos(\theta_1 - \theta_2) - M sin(\theta_1 - \theta_2)[l_1 \dot{\theta_1^2} cos(\theta_1 - \theta_2) + l_2 \dot{\theta_2^2}]}{l_1 [1 - M cos^2(\theta_1 - \theta_2)]} \tag{2.11}$$

$$\ddot{\theta_2} = \frac{-g sin\theta_2 + cos(\theta_1 - \theta_2)[M l_2 \dot{\theta_2^2} sin(\theta_1 - \theta_2) + g sin\theta_1] + l_1 \dot{\theta_1^2} sin(\theta_1 - \theta_2)}{l_2 [1 - M cos^2(\theta_1 - \theta_2)]}, \tag{2.12}$$

where $M = \frac{m_2}{m_1 + m_2}$.

2.3 Model

Using equations (2.11) and (2.12), we can establish a set of first order differential equations which will be the model we use for the double pendulum. Then,

$$\tau_1 = \theta_1\,,\ \tau_2 = \theta_2\,,\ \tau_3 = \dot{\theta_1}\,,\ \tau_4 = \dot{\theta_2} \tag{2.13}$$

$$\dot{\tau_1} = \dot{\theta_1} = \tau_3\,,\ \dot{\tau_2} = \dot{\theta_2} = \tau_4\,,\ \dot{\tau_3} = \ddot{\theta_1}\,,\ \dot{\tau_4} = \ddot{\theta_2}. \tag{2.14}$$

Substituting components from equations (2.13) and (2.14) into equations (2.11) and (2.12), the model becomes

$$\dot{\tau_1} = \tau_3$$

$$\dot{\tau_2} = \tau_4$$

$$\dot{\tau_3} = \frac{-gsin\tau_1 + Mgsin\tau_2 cos(\tau_1 - \tau_2) - Msin(\tau_1 - \tau_2)[l_1\tau_3^2 cos(\tau_1 - \tau_2) + l_2\tau_4^2]}{l_1[1 - Mcos^2(\tau_1 - \tau_2)]} \text{ and}$$

$$\dot{\tau_4} = \frac{-gsin\tau_2 + cos(\tau_1 - \tau_2)[Ml_2\tau_4^2 sin(\tau_1 - \tau_2) + gsin\theta_1] + l_1\tau_3^2 sin(\tau_1 - \tau_2)}{l_2[1 - Mcos^2(\tau_1 - \tau_2)]}.$$

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Methodology

In this chapter, we explain the various methods used to generate or extract pseudorandom bit sequences from the model we derived. We also discuss how those sequences are tested for randomness.

3.1 Sequence Generation

Using MATLAB, a double pendulum is generated using the model we previously derived along with a set of parameters applied over a period of time. The parameters are $m_1, m_2, l_1, l_2, \theta_1, \theta_2$, and g. MATLAB solves the model for θ_1, θ_2, $\dot{\theta_1}$, and $\dot{\theta_2}$ as functions of time using the function *ode*45. The initial conditions for sequence generation are the same for each sequence, where $\theta_1(0) = \theta_1$, $\theta_2(0) = \theta_2$, $\dot{\theta_1}(0) = 0$, and $\dot{\theta_2}(0) = 0$. Once the double pendulum has been generated, the x and y positions of the first and second masses are extracted using equations (2.1) and (2.2). Other information, such as angular speed, is extracted as necessary. Six different methods were utilized to generate sequences: linear, radial, derivative comparison, height comparison, energy comparison, and image. Preliminary testing indicated that the energy comparison and image methods were most effective in generating pseudorandom sequences, and hence results analysis will focus primarily on these two methods.

3.1.1 Linear Method

For the linear method, we only consider the motion of the second mass and where it crosses a set of chosen lines. Two horizontal lines and two vertical lines are chosen such that the second mass will cross them in its motion for a given time period. The lines are given values, either 1 or 0, such that when the second mass crosses that line, the associated value is appended to the sequence to be generated. However, this introduces elevated numbers of double values such as "00" or "11" into the sequence. Given a random sequence, we would expect each bit to appear with probability $1/2$, and such repeated bits, typically called runs of length 2, to appear with probability $1/4$. To overcome this problem, a second pendulum with different parameters is generated and its motion is analyzed simultaneously with the

first pendulum. The value of the bits generated by crossing the lines can also be altered to be different from that of the first pendulum, but the position of the lines remains the same. Lines were established at x = -1.1, x = 1.1, y = 0, and y = -1.5 for sequence generation. The code checks for a crossing condition by progressing through the x and y positions of the pendulums, comparing the present and previous values for crossing the lines in either direction. See Figure 3.1.

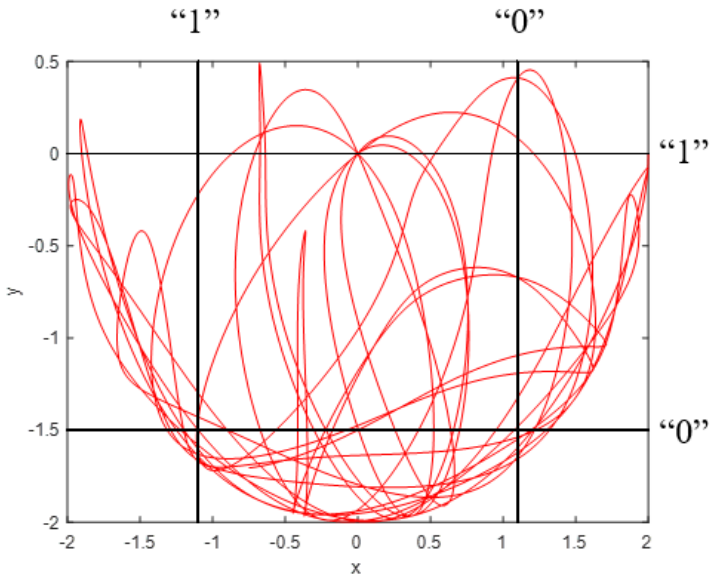

Figure 3.1. Linear Method Diagram

3.1.2 Radial Method

For the radial method, the position of the second mass is computed as a vector originating at the origin using the equation

$$d = \sqrt{x_2^2 + y_2^2} \tag{3.1}$$

Similar to the linear method, the sequence is generated based on the second mass crossing over circles of chosen radii. Equation (3.1) is used to calculate the present and previous values of the position of the second mass, then the values are compared to determine if a circle has been crossed. The circles are assigned values, either 1 or 0, such that the associated value is appended to the sequence when the mass crosses the circle. To add more variability, the values assigned change based on the quadrant, I-IV, in which the crossing

occurs. Circles with radii 1 and 1.9 centered at the origin were established for sequence generation. See Figure 3.2.

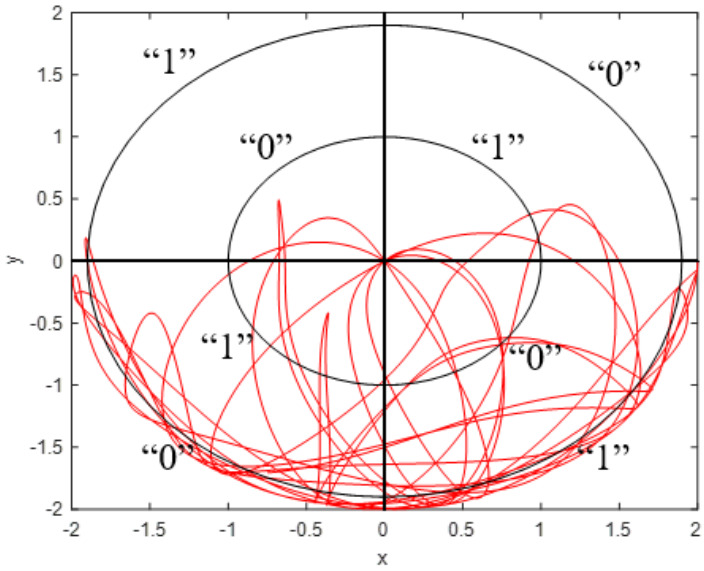

Figure 3.2. The Double Pendulum

3.1.3 Derivative Comparison Method

In the derivative comparison method, the instantaneous angular speed of the first and second masses, $\dot{\theta}_1$ and $\dot{\theta}_2$, is calculated from two generated pendulums using the respective equations derived previously. The angular speed for the second masses are compared. If the second mass of the second pendulum has a higher angular speed than the second mass of the first pendulum, then we check the angular speed of the first mass of the first pendulum. If the angular speed of the first mass is positive or 0, the next value in the sequence is a 1, otherwise 0. If the second mass has a lower angular speed, then the opposite occurs where if the angular speed of the first mass is negative, the next value in the sequence is a 1, otherwise 0. This comparison is made for every point in the generated pendulums.

3.1.4 Height Comparison Method

The height comparison method works in the same way as the derivative comparison method except that instead of angular speeds, it compares y positions only.

3.1.5 Energy Comparison Method

The energy method uses separated variants of the kinetic and potential energy equations to determine the kinetic and potential energies of both masses of a generated pendulum. We then compute the average kinetic energy and average potential energy for each of the masses. If the kinetic energy of the second mass crosses its average kinetic energy and the kinetic energy of the first mass is greater than its average kinetic energy, then a value of 1 is appended to the sequence. If the kinetic energy of the first mass is less than its average, then a value of 0 is appended to the sequence. The same comparison is made simultaneously using the potential energies instead.

3.1.6 Image Method

For the image method, the motion of the second mass of the double pendulum is plotted and a binary image is made from a chosen window on the plot. The sequence is then the binary image, read from left to right, top to bottom. The window in which the image is generated is maximized in order to achieve longer sequences. Since the image is always the same size, all sequences generated using the image method are the same length. The window used in this method has x coordinate limitations of -0.1 to 0.1 and y coordinate limitations of -.51 to -.49 in order to acquire the most motion in one image.

3.1.7 Limitations

Each method has limitations with respect to the length of the sequences they generate, the composition of the sequences, as well as the time required to generate a sequence of a desired length. The energy method produces undesirable results when there is little to no chaotic motion; the motion appears similar to that of a normal pendulum, which produces relatively non-random sequences. The image method also produces undesirable results when there is little to no chaotic motion. The window may not contain any motion, resulting in a sequence consisting entirely of 0's.

3.2 Experimental Setup

Based on sample size and the maximum input size recommendations from NIST, we use a significance level, α, of 0.01 to generate 100 sequences that are at least one million bits in

length using the energy method and the image method. The sequences are generated and tested on an ASUS Model GL533VD laptop with the following specifications:

Operating System:	Windows 10 Home
System Type:	64-bit
CPU:	Intel i7-7700HQ (2.8 GHz, 4 cores)
RAM:	1x16 GB @ 2400 MHz
Integrated Graphics:	Intel HD Graphics 630
GPU:	NVIDIA GeForce GTX 1050 (4 GB GDDR5)

The energy method and image method receive more focus than the other methods due to the fact that they consistently generate long enough sequences, at least 1 million bits in length, in the least amount of time. For example, using the linear method, a double pendulum that ran over a time span of 400000 seconds would be required to make a sequence of at least one million bits. Generating this double pendulum takes considerably longer than using the energy method with a double pendulum generated over a time span of 200000 seconds which also makes a sequence of at least one million bits. The experimental setup involves generating 100 sequences using each method and varying one parameter while keeping the others constant. The default parameters are as follows:

$$\begin{array}{cc} m_1 = 1kg & m_2 = 1kg \\ l_1 = 1m & l_2 = 1m \\ \theta_1 = \frac{\pi}{2}rad & \theta_2 = \frac{\pi}{2}rad \\ g = 9.8\ m/s^2 & \end{array}$$

We vary each mass from 0.5 to 1.5 kg, vary each length from 0.5 to 1.5 m, and vary each angle from $-\pi/2$ to $\pi/2$ rad, with 100 points in each range for 100 different parameters for each variable, therefore creating 100 different sequences for each variable, per method. Gravity remains constant for all sequences generated.

3.3 Randomness Testing

In order to determine if the sequences generated are random and viable for cryptographic applications, we utilize a statistical test suite developed by the NIST, [3]. The suite consists

of fifteen different tests to evaluate the randomness of the sequences we generated. Passing or failing a single test does not necessarily imply that the sequence being tested is random or non-random, but passing more tests allows us to state that there is sufficient statistical evidence to infer that the sequence is random. The test suite was coded in Python from [4] and verified using the examples from [3].

3.3.1 Monobit Frequency Test

The Monobit Frequency Test determines if the proportion of 1's to 0's in a sequence is close to $1/2$, which is the same proportion for a truly random sequence. In this sense, the sequence is balanced since it contains an equal number of 1's and 0's. This test is the first test conducted in the battery of tests because failing it implies failure of all subsequent tests.

3.3.2 Block Frequency Test

The Block Frequency Test is similar to the Monobit Frequency Test except that instead of determining the proportion of 1's and 0's in the entire sequence, it examines the proportions in blocks of a specified length M. The frequency of 1's should be $M/2$ assuming the sequence is random. For our tests, we use $M = 128$.

3.3.3 Runs Test

The Runs Test determines the total number of runs in the sequence, where a run is a number of 1's bounded by 0's or vice versa. The test determines if the sequences has too many or too few runs in comparison to a random sequence. In a random sequence, a 1 or a 0 has a probability of appearing of $1/2$, so the probability of having a run of length n is $1/2^n$. Hence, in a truly random sequence, a pattern of runs should not exist.

3.3.4 Longest Run of Ones Test

The Longest Run of Ones Test determines the longest run of 1's in a block of length M in the sequence. The test is separated into three parts based on the length of the block, where $M = 8$, 128, or 10^4. This test compares the longest run of 1's to what would be expected from a truly random sequence of the same length given the block size.

3.3.5 Binary Matrix Rank Test

The Binary Matrix Rank Test determines the rank of matrices generated from the sequence, which identifies linear dependence between parts of the sequence which can be compared to that of a truly random sequence of the same length. Based on the NIST recommendations for this test, the matrices generated have 32 rows and 32 columns in our tests.

3.3.6 Spectral Test

The Spectral Test essentially conducts a Fourier transform on the sequence and determines if a periodic nature exists, which would indicate that the sequence is non-random.

3.3.7 Non-overlapping Template Matching Test

The Non-overlapping Template Matching Test uses an aperiodic template sequence and steps through the sequence to determine if the template exists within it. If the template does not exist, the template moves by one bit. If the template matches, it skips the length of the template to the next set of bits in the sequence, hence the non-overlapping nature of the test. Too many or too few occurrences of the template indicate that the sequence is non-random. For our tests, we used the aperiodic template 000000001 based on the input size recommendations from section 2.7.7 of [3].

3.3.8 Overlapping Template Matching Test

The Overlapping Template Matching Test is the same as the Non-overlapping Template Matching Test except that the template will always move to the next bit regardless of matching. For our tests, we use the template 111111111 with $K = 5, M = 1032$ based on the input size recommendations from section 2.8.7 of [3], where K is the number of degrees of freedom and M is the length in bits of a substring of the test sequence.

3.3.9 Maurer's "Universal Statistical" Test

Maurer's "Universal Statistical" Test determines if the sequence can be compressed given a block of length L. A compressible sequence would be non-random. For our tests, we used $L = 7$ and $Q = 1280$ based on the input size recommendations from section 2.9.7 of [3], where Q is the number of blocks in an initialization sequence.

3.3.10 Linear Complexity Test

The Linear Complexity Test determines the length of a linear feedback shift register (LFSR) that would be needed to generate the sequence. If the LFSR is too short, the sequence is non-random.

3.3.11 Serial Test

The Serial Test determines the number of overlapping patterns across the whole sequence. If a sequence is random, then any given pattern is as likely to appear as a different pattern in the sequence.

3.3.12 Approximate Entropy Test

The Approximate Entropy Test is the same as the Serial Test, except that it compares the frequency of overlapping patterns of adjacent lengths against those of a random sequence.

3.3.13 Cumulative Sums Test

The Cumulative Sums Test determines how much the partial sums of an adjusted sequence deviate from 0. The sequence is adjusted so that 1's remain 1's and 0's become -1's. A sequence that is random will have few deviations from 0, whereas a non-random sequence will have large deviations.

3.3.14 Random Excursions Test

The Random Excursions Test is similar to the Cumulative Sums Test, except that it examines the frequency of occurrence for 8 different states, -4, -3,..., 3, 4 , in a random walk within the sequence. Deviations from the expected distribution of frequencies given a random sequence would indicate that the sequence is non-random.

3.3.15 Random Excursions Variant Test

The Random Excursions Variant Test is the same as the Random Excursions Test, except that it examines the frequency of occurrence for 18 different states vice 8 different states, which are -9, -8, ..., 8, 9.

CHAPTER 4:
Results and Analysis

In this chapter, we present and analyze the results of the NIST tests for the energy and the image sequence generation methods. Based on section 4.2 of [3], the results are interpreted using the proportion of sequences that pass the tests and checking for uniformity of p-values across the tests.

4.1 Energy Method Results and Analysis

As mentioned in the previous chapter, 100 different sequences were generated over varying parameters for mass, arm length, and angle for a total of 600 sequences. The following subsections discuss the results and analysis of the NIST tests for the energy method. For figures showing the pass proportions for each of the tests, the green line represents 1-α and the red lines represent the confidence interval as determined using section 4.2.1 of [3]. In general, sequences generated using the energy method passed 7 out of the 15 tests, but only the Block Frequency Test was passed with proportions exhibiting statistical significance in 3 out of the 6 different variations. All of the sequences generated by the energy method have a ratio of 1's to 0's of about 50%, so this justifies why the sequences passed the Block Frequency Test in higher proportions. None of the sequences passed the Runs Test because the sequences contain too many runs in comparison to that of a truly random sequence. The runs are short, usually two bits in length, but are frequent. The frequency of these runs is likely due to the energy method's bit formation based on the average energy.

4.1.1 First Mass Variation

The first mass was varied from 0.5 kg to 1.5 kg over 100 values for a total of 100 sequences generated using the energy method. We see that all of the sequences passed the Block Frequency Test, but this is the only test passed that is statistically significant. We also see that the distribution of p-values is heavily skewed to values between 0 and 0.1, but is otherwise uniformly distributed. The results are shown in Figures 4.1 and 4.2.

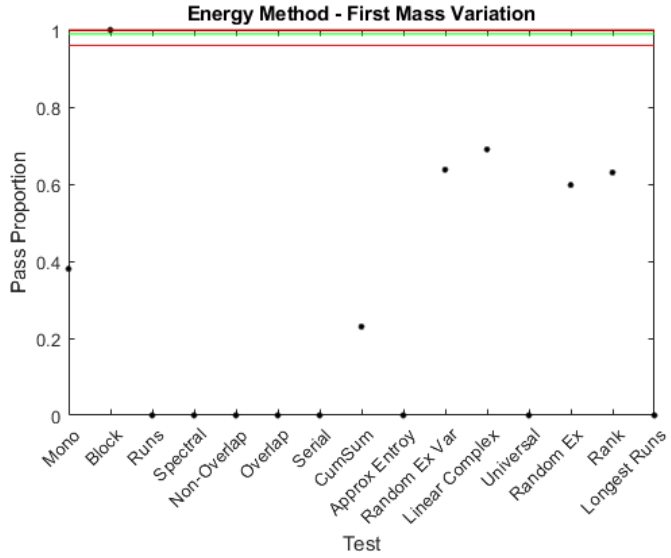

Figure 4.1. Energy Method - First Mass Variation NIST Test Results

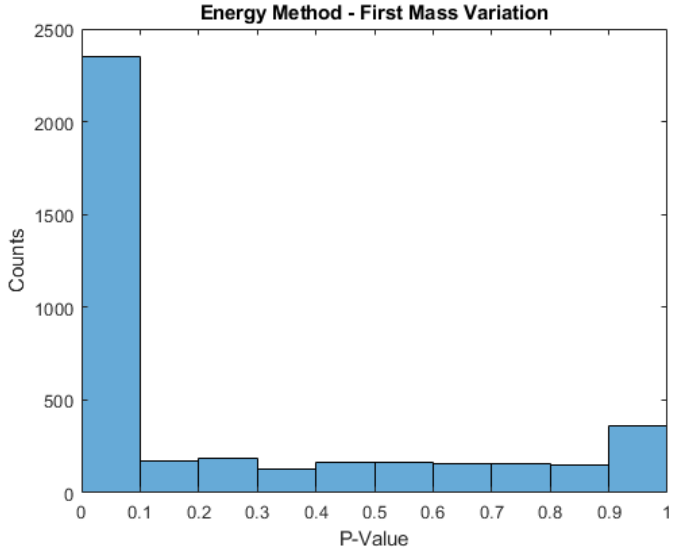

Figure 4.2. Energy Method - First Mass Variation P-Value Distribution

4.1.2 Second Mass Variation

The second mass was varied from 0.5 kg to 1.5 kg over 100 values for a total of 100 sequences generated using the energy method. Similar to the first mass variation, the Block

Frequency Test is the only test where all sequences passed. A fewer percentage of these sequences passed the Monobit Frequency Test in comparison to those sequences generated by first mass variation. Again, the p-values are heavily skewed to between 0 and 0.1 as with the first mass variation. The results are shown in Figures 4.3 and 4.4.

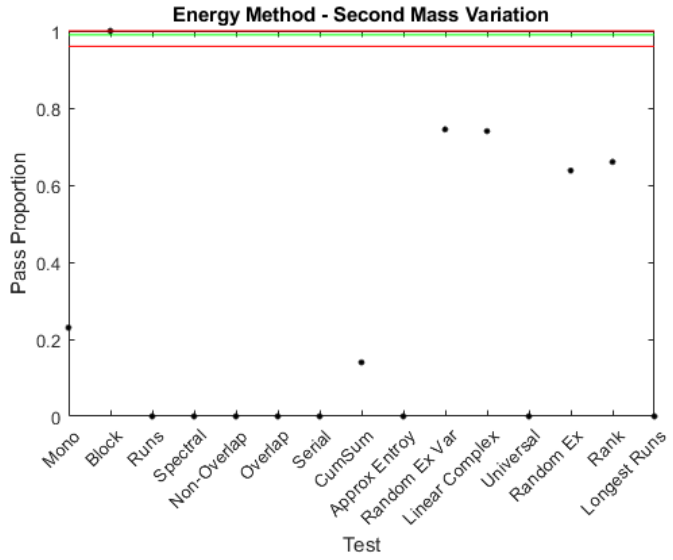

Figure 4.3. Energy Method - Second Mass Variation NIST Test Results

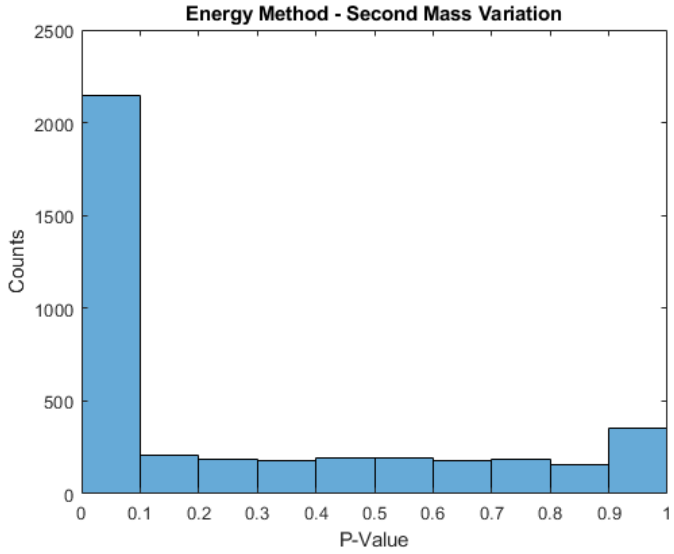

Figure 4.4. Energy Method - Second Mass Variation P-Value Distribution

4.1.3 First Length Variation

The length of the first arm was varied from 0.5 m to 1.5 m over 100 values for a total of 100 sequences generated using the energy method. With the first length variation, none of the tests pass with statistical significance. Substantially fewer sequences passed the Monobit Frequency Test, but more sequences passed the Random Excursion tests. From the p-value distribution, we see that overall fewer sequences failed, but the distribution is skewed to between 0 and 0.1. The results are shown in Figures 4.5 and 4.6.

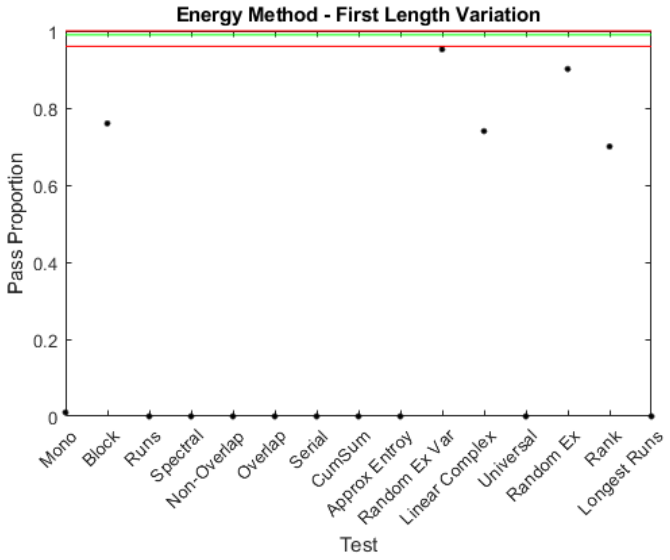

Figure 4.5. Energy Method - First Length Variation NIST Test Results

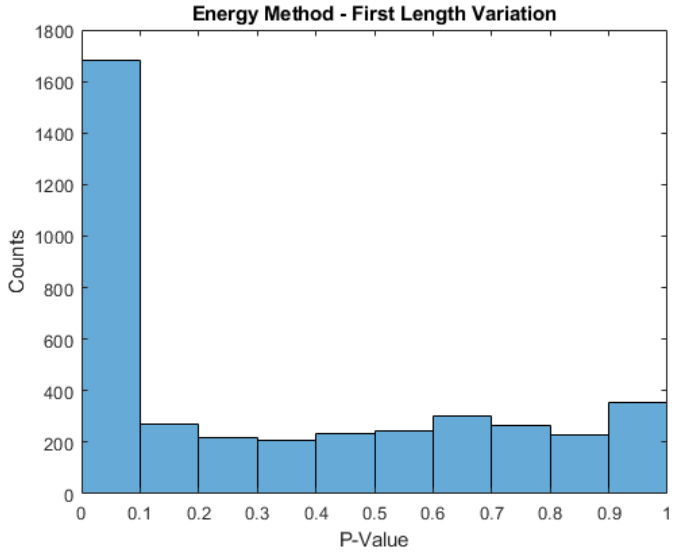

Figure 4.6. Energy Method - First Length Variation P-Value Distribution

4.1.4 Second Length Variation

The length of the second arm was varied from 0.5 m to 1.5 m over 100 values for a total of 100 sequences generated using the energy method. By varying the second length, the sequences passed the Block Frequency Test with statistical significance. Fewer sequences passed the random excursion tests in comparison to varying the first length. Approximately 35% of the sequences passed the Monobit Frequency Test and approximately 20% of the sequences passed the Cumulative Sums Test, whereas no sequence passed either of those tests for the first length variation. The p-values are skewed to between 0 and 0.1 with a higher count than the first length. The results are shown in Figures 4.7 and 4.8.

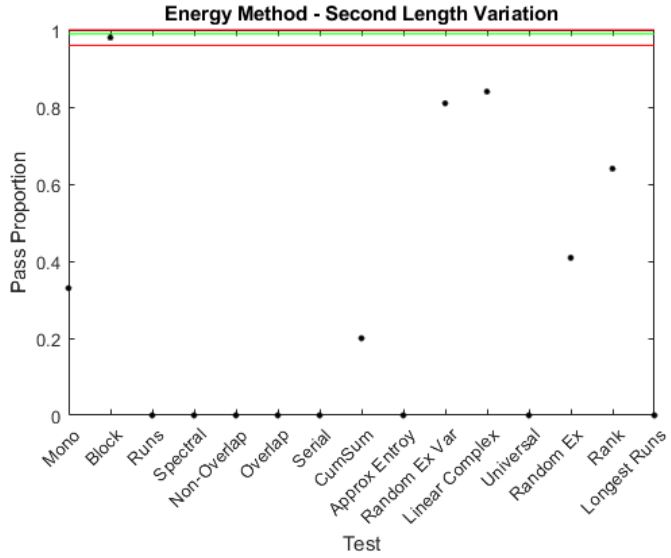

Figure 4.7. Energy Method - Second Length Variation NIST Test Results

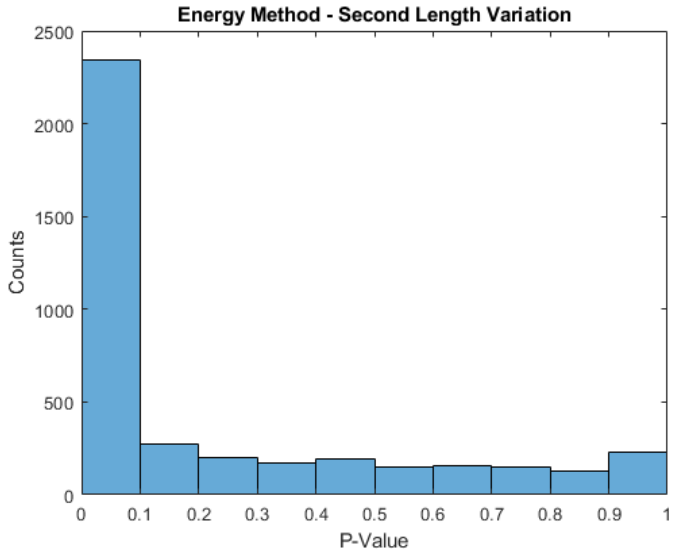

Figure 4.8. Energy Method - Second Length Variation P-Value Distribution

4.1.5 First Angle Variation

The angle between the first arm and the vertical was varied from $-\pi/2$ rad to $\pi/2$ rad over 100 values for a total of 100 sequences generated using the energy method. The sequences

generated by varying the first angle did not pass any of the tests with statistical significance. The results are shown in Figures 4.9 and 4.10.

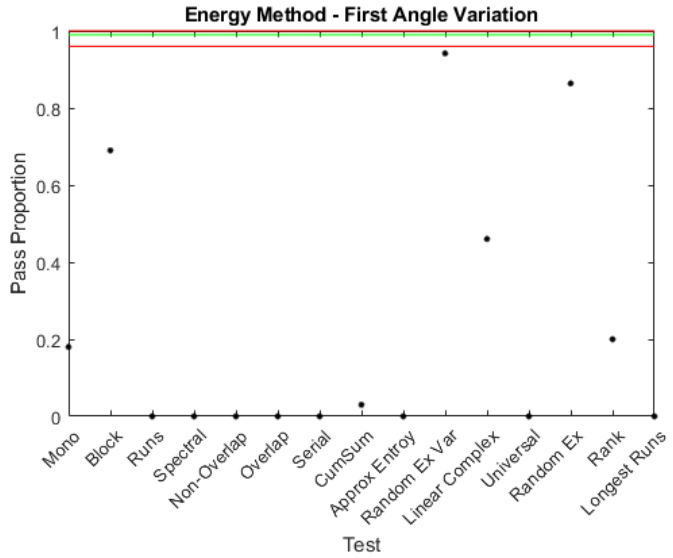

Figure 4.9. Energy Method - First Angle Variation NIST Test Results

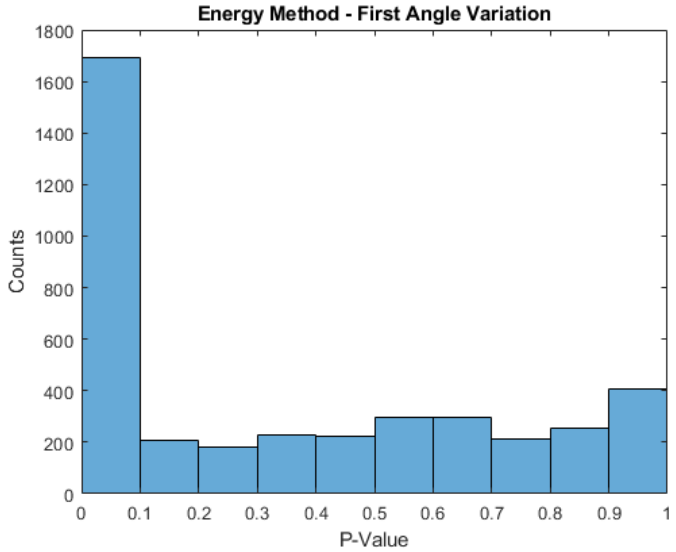

Figure 4.10. Energy Method - First Angle Variation P-Value Distribution

4.1.6 Second Angle Variation

The angle between the second arm and the vertical was varied from $-\pi/2$ rad to $\pi/2$ rad over 100 values for a total of 100 sequences generated using the energy method. The results are similar to the first angle variation and are shown in Figures 4.11 and 4.12.

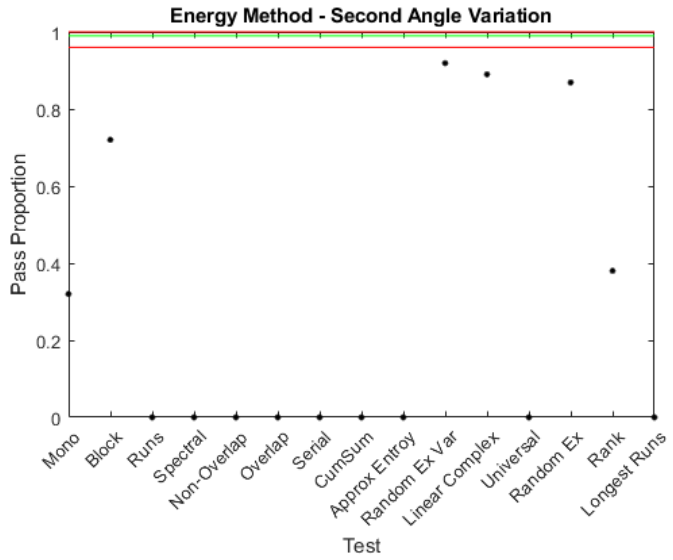

Figure 4.11. Energy Method - Second Angle Variation NIST Test Results

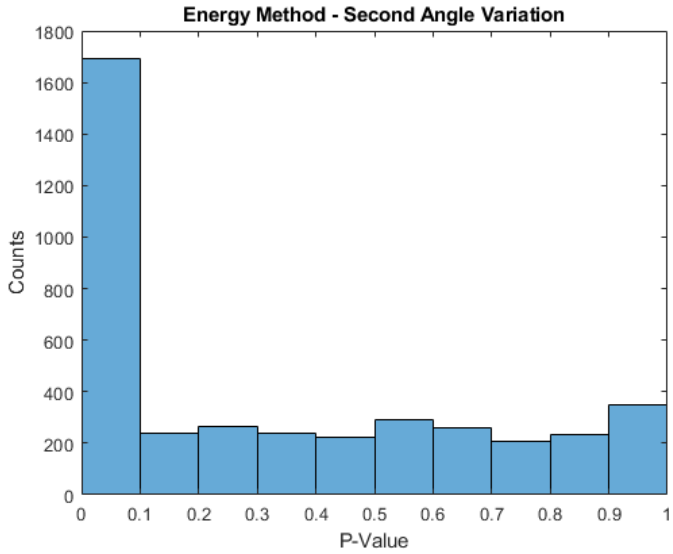

Figure 4.12. Energy Method - Second Angle Variation P-Value Distribution

4.1.7 Linear Complexity

Using the Berlekamp-Massey algorithm Python code found in [5], the linear complexity profile is determined for the energy method. Although not all sequences passed the linear complexity test, the sequences exhibit excellent linear complexity as shown in Figure 4.13.

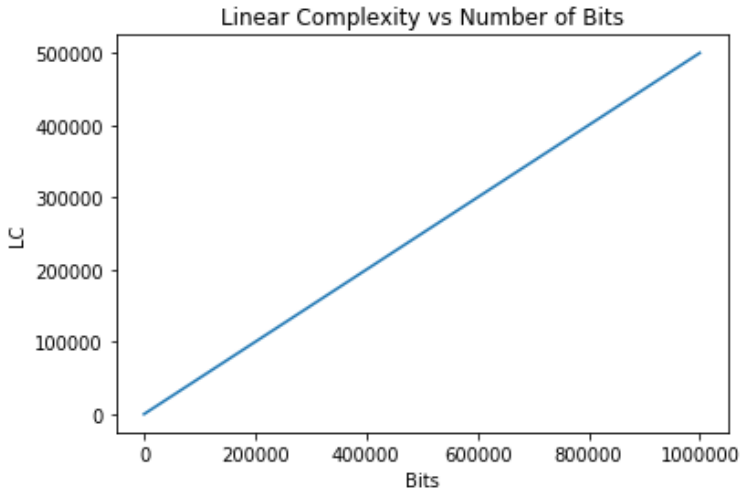

Figure 4.13. Energy Method - Linear Complexity Profile

4.2 Image Method Results and Analysis

As mentioned in the previous chapter, 100 different sequences were generated over varying parameters for mass, arm length, and angle for a total of 600 sequences. The following subsections discuss the results and analysis of the NIST tests for the image method.

4.2.1 First Mass Variation

The first mass was varied from 0.5 kg to 1.5 kg over 100 values for a total of 100 sequences generated using the image method. In comparison to the energy method, a much smaller proportion of sequences passed the tests, and none of them passed with statistical significance. The distribution of p-values for the image method is similar to the energy method in that most of the p-values are between 0 and 0.1. However, we see that the rest of the distribution is not uniform and gradually increases towards the range of 0.9 to 1. The results are shown in Figures 4.14 and 4.15.

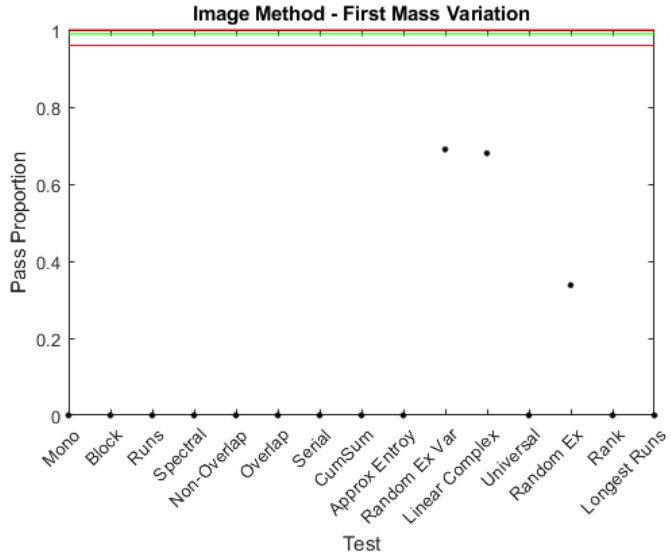

Figure 4.14. Image Method - First Mass Variation NIST Test Results

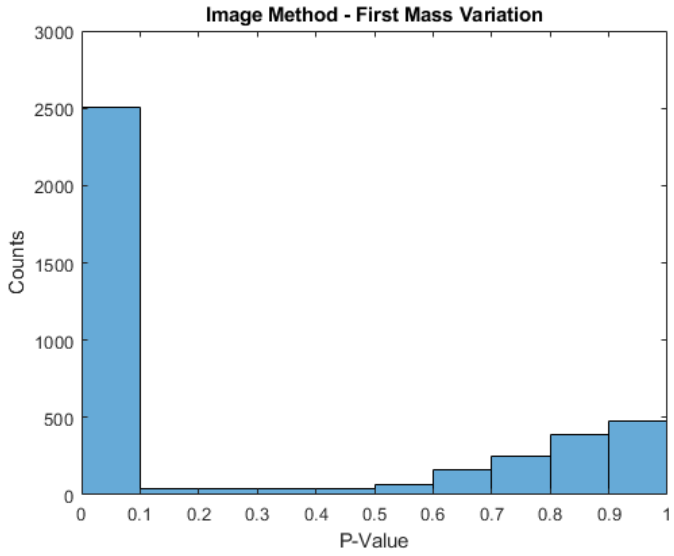

Figure 4.15. Image Method - First Mass Variation P-Value Distribution

4.2.2 Second Mass Variation

The second mass was varied from 0.5 kg to 1.5 kg over 100 values for a total of 100 sequences generated using the image method. The results are very similar to those for the

first mass variation and are shown in Figures 4.16 and 4.17.

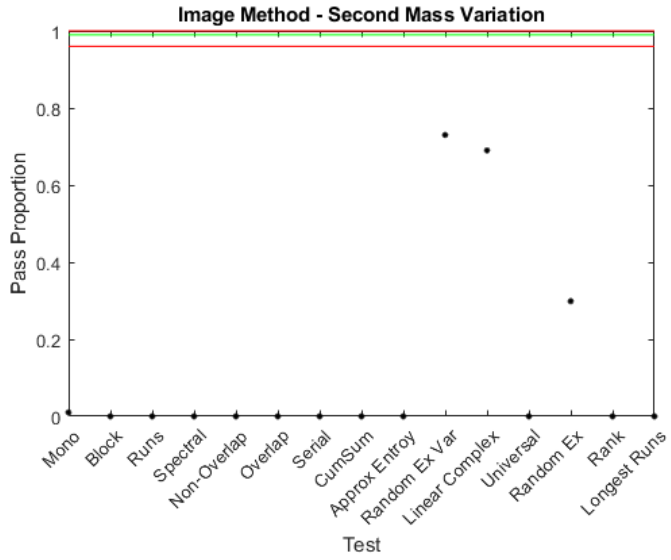

Figure 4.16. Image Method - Second Mass Variation NIST Test Results

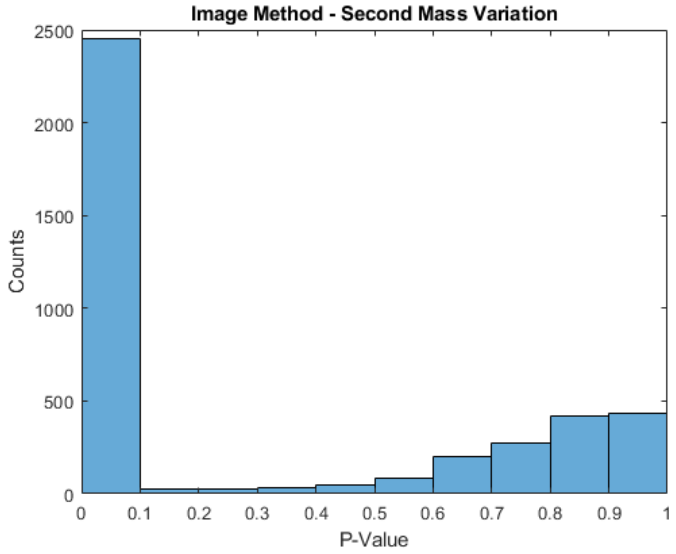

Figure 4.17. Image Method - Second Mass Variation P-Value Distribution

4.2.3 First Length Variation

The length of the first arm was varied from 0.5 m to 1.5 m over 100 values for a total of 100 sequences generated using the image method. The results are similar to those for mass variation, but less than 10% of the sequences passed the Binary Matrix Rank Test and less than 5% passed the Spectral Test. The results are shown in Figures 4.18 and 4.19.

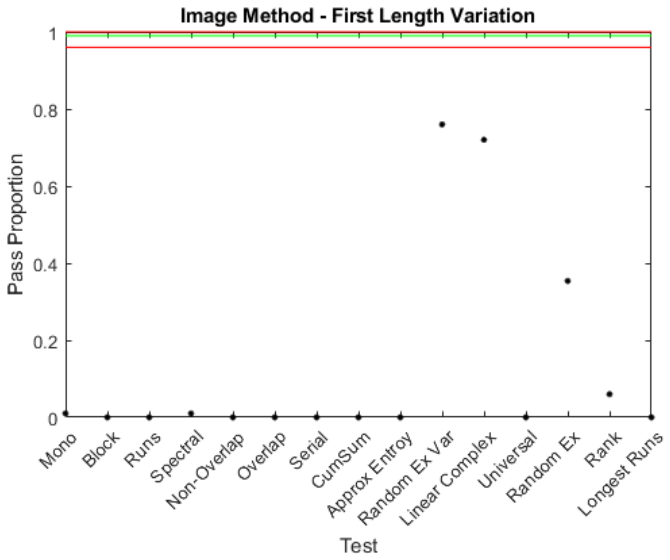

Figure 4.18. Image Method - First Length Variation NIST Test Results

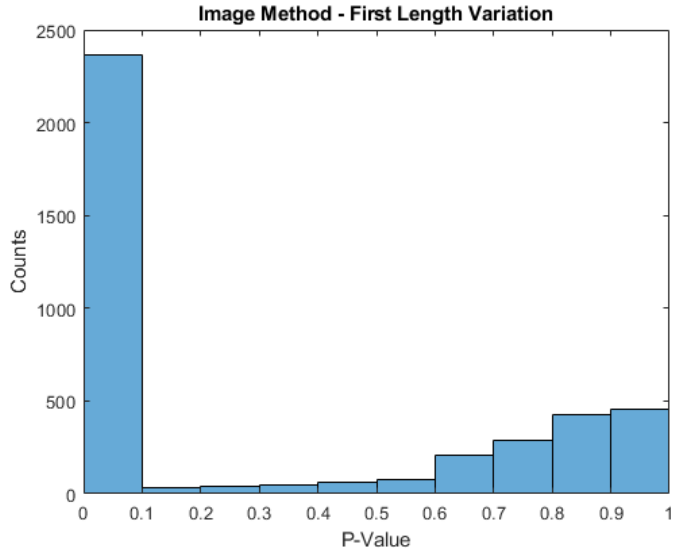

Figure 4.19. Image Method - First Length Variation P-Value Distribution

4.2.4 Second Length Variation

The length of the second arm was varied from 0.5 m to 1.5 m over 100 values for a total of 100 sequences generated using the image method. The results are similar to those for the first length variation except that fewer sequences passed the Binary Matrix Rank Test and slightly more sequences passed the Spectral Test. The results are shown in Figures 4.20 and 4.21.

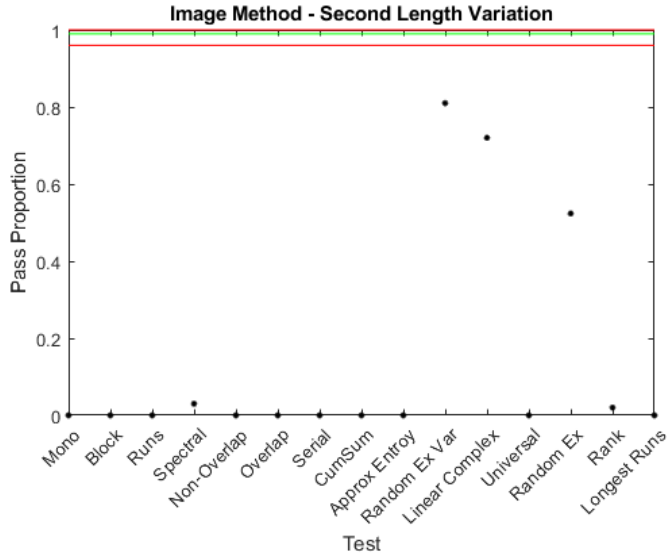

Figure 4.20. Image Method - Second Length Variation NIST Test Results

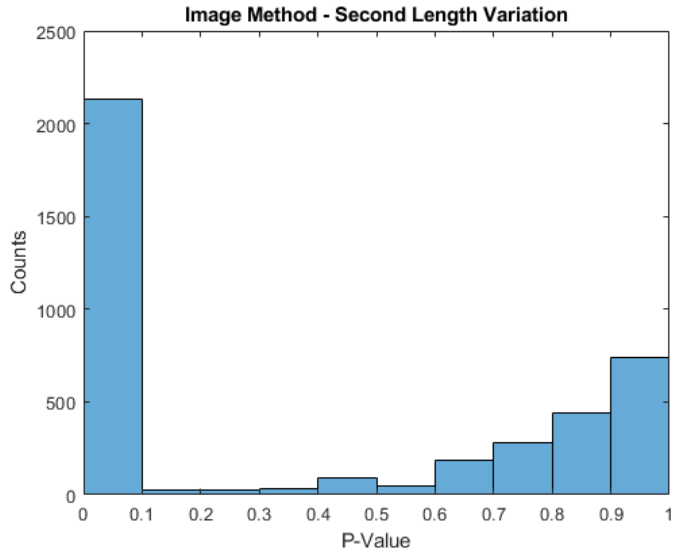

Figure 4.21. Image Method - Second Length Variation P-Value Distribution

4.2.5 First Angle Variation

The angle between the first arm and the vertical was varied from $-\pi/2$ rad to $\pi/2$ rad over 100 values for a total of 100 sequences generated using the image method. Fewer sequences

passed the tests in comparison to the mass or length variations overall. The results are shown in Figures 4.22 and 4.23.

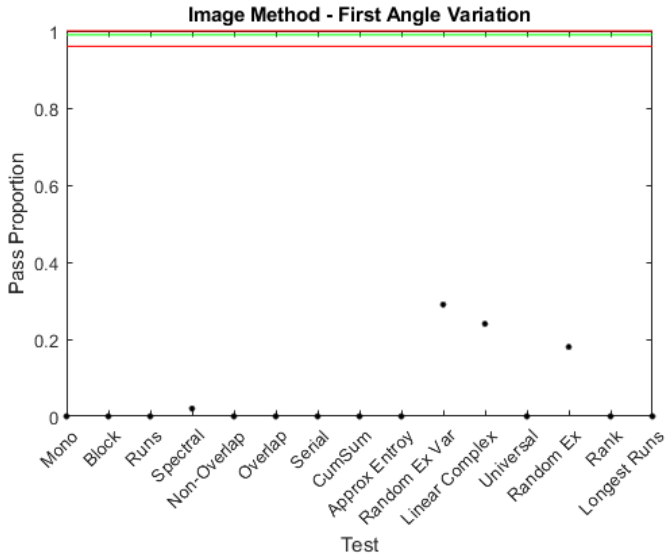

Figure 4.22. Image Method - First Angle Variation NIST Test Results

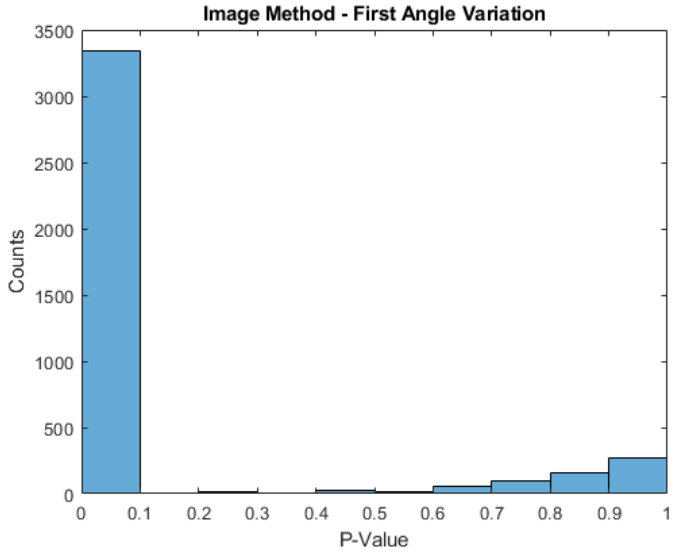

Figure 4.23. Image Method - First Angle Variation P-Value Distribution

4.2.6 Second Angle Variation

The angle between the second arm and the vertical was varied from $-\pi/2$ rad to $\pi/2$ rad over 100 values for a total of 100 sequences generated using the image method. More sequences passed the Random Excursion Tests, but almost none of the sequences passed the Linear Complexity Test. The results are shown in Figures 4.24 and 4.25.

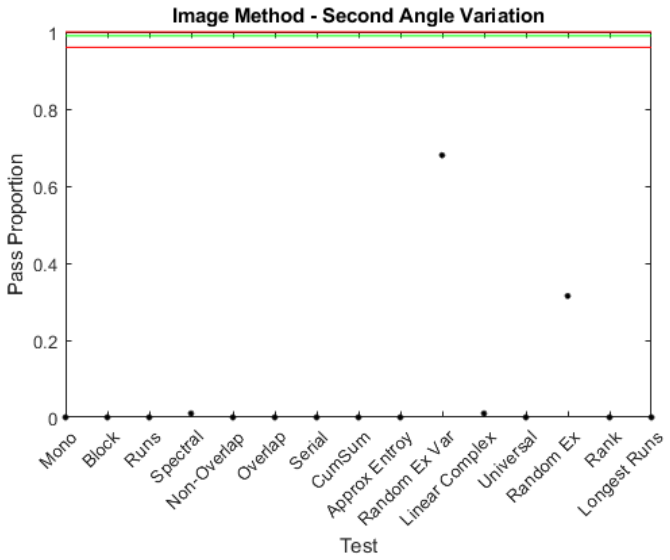

Figure 4.24. Image Method - Second Angle Variation NIST Test Results

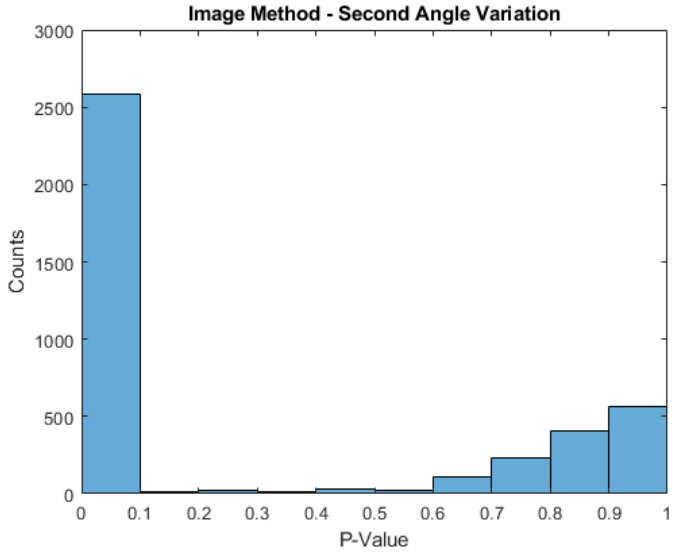

Figure 4.25. Image Method - Second Angle Variation P-Value Distribution

4.2.7 Linear Complexity

Using the Berlekamp-Massey algorithm Python code found in [5], the linear complexity profile is determined for the image method. In contrast to the energy method, while some sequences passed the linear complexity test, the majority of sequences generated by the image method do not exhibit linear complexity as shown in Figure 4.26.

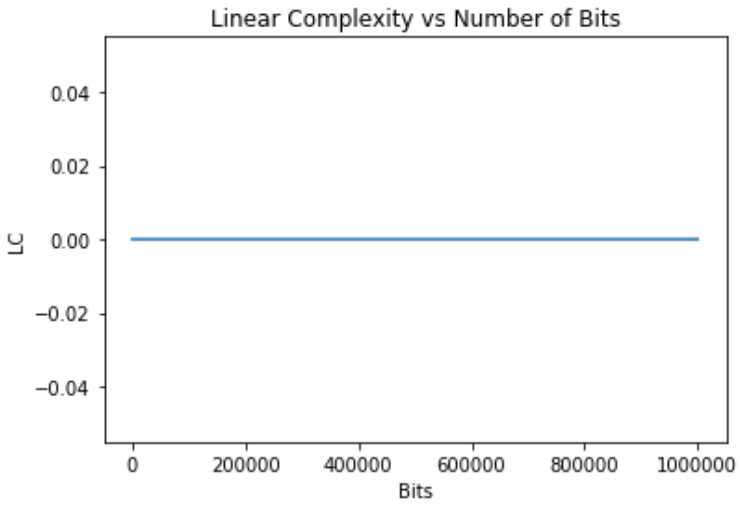

Figure 4.26. Image Method - Linear Complexity Profile

CHAPTER 5:
Conclusion and Future Work

This thesis sought to exploit the chaotic motion of a double pendulum to generate pseudorandom sequences. The results show that, unfortunately, the methods we used were inadequate to produce sequences that could be classified as pseudorandom based on the NIST tests. The variations made to each of the methods yielded only one test being passed with statistical significance in one of the methods, which is insufficient to say that the sequences were random without passing other tests in the suite. Additionally, the distribution of p-values was not uniform for any of the variations in either the energy method or the image method, and was skewed towards failure. Future work could consist of identifying where the methods used in this thesis failed or how they could be improved so as to generate better sequences. Additional methods to generate sequences from the double pendulum should be developed and analyzed. Consideration should also be given to other chaotic systems besides the double pendulum to determine if the same or better results can be achieved from them.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX: MATLAB and Python Code

The appendix includes MATLAB code used to define the double pendulum model, generate the binary sequences, and analyze the results. Also included is the Python code used to test the sequences. The original overall test code is supplied by Alex Gutzler and can be found in [6]; it has been modified for this thesis. The Python code for the NIST tests can be downloaded from [4].

A.1 Double Pendulum MATLAB Function

The following code is a MATLAB function containing the model derived in Chapter 2 to be solved with the *ode*45 function.

```
%Equations of motion setup for double pendulum
%Desc: Defines first-order differential equations of motion for the double
%pendulum based on input parameters time (t), initial positions/velocities
%(y), mass (m), and length (l). Time serves only for evaluation via ode45
%and is not actually used in this function. Initial positions/velocities
%must be in the form of a 1x4 vector, where y(1) = initial angle of first
%mass (theta_1, in radians), y(2) = initial angle of second mass (theta_2,
%in radians), y(3) = initial angular velocity of first mass (usually 0),
%and y(4) = initial angular velocity of second mass (usually 0). Mass must
%be a 1x2 vector containing the mass values for the first and second masses
%of the pendulum (m(1) and m(2) respectively, in kg). Length must be a 1x2
%vector containing the length between the origin and the first mass (l(1),
%in meters) and the length between the first mass and the second mass
%(l(2), in meters). The output is a 4x1 vector for use in the ode45
%function.
function dydt = doublepend(t,y,m,l)
dydt = zeros(4,1);
M = m(1)+m(2);
a = sin(y(1)-y(2));
b = cos(y(1)-y(2));
g = 9.8; %gravity, m/s^2
```

```
y1 = y(3);
y2 = y(4);
y3 = (-M*g*sin(y(1))+m(2)*g*sin(y(2))*b-m(2)*y(3)*y(3)*l(1)*a*b-m(2)...
    *l(2)*y(4)*y(4)*a)/(l(1)*M-m(2)*l(1)*b^2);
y4 = (-g*sin(y(2))*M+m(2)*l(2)*y(4)*y(4)*a*b+M*g*sin(y(1))*b+l(1)*y(3)...
    *y(3)*a*M)/(l(2)*M-m(2)*l(2)*b^2);
dydt = [y1 y2 y3 y4]';
end
```

A.2 Sequence Generation MATLAB Code

The following MATLAB code uses the model function to first generate a pendulum based on the input parameters, then uses the energy and image method to generate binary sequences. The generated binary sequences are saved as text files to be tested using the Python code.

```
clear
close all
clc
%Base Parameters
%NOTE: Base parameters must be commented out as necessary for testing
m = [1 1]; %Mass vector in kg
l = [1 1]; %Length vector in meters
g = 9.8; %Gravity in m/s^2
th1 = pi/2; %Initial theta_1 in radians
th2 = pi/2; %Initial theta_2 in radians
%Parameters to be tested
%th1 = linspace(-pi/2,pi/2,100);
%th2 = linspace(-pi/2,pi/2,100);
%NOTE: For the following parameters, the vectors l and m must be defined
%in the loop.
%m1 = linspace(0.5,1.5,100);
%m2 = linspace(0.5,1.5,100);
%l1 = linspace(0.5,1.5,100);
%l2 = linspace(0.5,1.5,100);
tspan = [0 25]; %Time vector in seconds, where first value is initial time
```

```
%and second value is final time.
opts = odeset('Refine',15,'RelTol',1e-7,'AbsTol',1e-5); %Options for ode45
%to refine the solution
for i = 1:length(th1) %Must be altered to match parameter to be tested
tic
%l = [l1(i) 1];
%m = [m1(i) 1];
%NOTE: The parameters of ode45 must be altered to match what is being
%tested. In this case, we are testing th1.
[t,y1] = ode45(@(t,y1) doublepend(t,y1,m,l),tspan,[th1(i) th2 0 0],opts);
%Evaluate equations of motion
x12=l(1)*sin(y1(:,1))+l(2)*sin(y1(:,2)); %Second mass x-position
y12=-l(1)*cos(y1(:,1))-l(2)*cos(y1(:,2)); %Second mass y-position
KE11= 0.5.*m(1).*y1(:,3).^2.*l(1).^2; %Kinetic energy of first mass
KE12 = m(2).*0.5.*(y1(:,3).^2.*l(1).^2+y1(:,4).^2.*l(2)^2+2.*y1(:,3)...
    .*y1(:,4).*l(1).*l(2).*cos(y1(:,1)-y1(:,2))); %Kinetic energy of
%second mass
PE11 = -m(1).*g.*l(1).*cos(y1(:,1)); %Potential energy of first mass
PE12 = - m(2).*g.*(l(1).*cos(y1(:,1))+l(2).*cos(y1(:,2))); %Potential
%energy of second mass
mu1 = mean(KE11); %Average kinetic energy of first mass
mu2 = mean(KE12); %Average kinetic energy of second mass
pu1 = mean(PE11); %Average potential energy of first mass
pu2 = mean(PE12); %Average potential energy of second mass
k = 1; %Initiate index for energy method
%ENERGY METHOD
for j = 2:length(KE12)
        if((KE12(j)<mu2 & KE12(j-1)>mu2) | (KE12(j)>mu2 & KE12(j-1)<mu2)...
                & KE11(j) > mu1)
            C(k) = 1;
            k = k+1;
        end
        if((KE12(j)<mu2 & KE12(j-1)>mu2) | (KE12(j)>mu2 & KE12(j-1)<mu2)...
                & KE11(j) < mu1)
            C(k) = 0;
```

```
            k = k+1;
        end
        if((PE12(j)<pu2 & PE12(j-1)>pu2) | (PE12(j)>pu2 & PE12(j-1)<pu2)...
                & PE11(j) > pu1)
            C(k) = 1;
            k = k+1;
        end
        if((PE12(j)<pu2 & PE12(j-1)>pu2) | (PE12(j)>pu2 & PE12(j-1)<pu2)...
                & PE11(j) < pu1)
            C(k) = 0;
            k = k+1;
        end
end
Cs = num2str(C); %Convert sequence from vector to string
Cs(isspace(Cs)) = ''; %Remove spaces
dlmwrite(['etht1-',num2str(i),'.txt'],Cs,'') %Save sequence as text
%IMAGE METHOD
figure('units','normalized','outerposition',[0 0 1 1]) %Maximize window
%size
plot(x12(1:floor(length(x12)/10)),y12(1:floor(length(y12)/10)),'k') %Plot
%motion of second mass
xlim([-.1 0.1]) %Window x-limits
ylim([-.51 -.49]) %Window y-limits
im = getframe(gca); %Capture image
close all
im = im.cdata(:,:,1); %Get first layer (all layers are the same)
cc = double(im); cc = cc == 0; %Convert image to type double, then invert
%image so that the black lines are 1's and the white spaces are 0's
s = size(cc); %Get size of image
C = reshape(cc,1,s(1)*s(2)); %Reshape image into a vector column by column,
%row by row as the sequence
Cs = num2str(C); %Convert sequence into string
Cs(isspace(Cs)) = ''; %Remove spaces
dlmwrite(['itht1-',num2str(i),'.txt'],Cs,'') %Save sequence as text
clc
```

```
Lines 79-84 determine the estimated time remaining to complete all 100
sequences; not required, but convenient
trun = round(toc);
perco = i;
perre = 100 - perco;
tremm = floor(perre*trun/60);
trems = perre*trun - 60*tremm;
fprintf('%3.1f %% complete, %3.0f min %3.0f sec remaining\n',...
    [perco tremm trems])
%Variables are cleared to save memory between loop iterations
clear t y1 x12 y12 KE11 PE11 KE12 PE12 mu1 mu2 pu1 pu2 k C Cs im s cc...
    trun perco perre tremm trems
end
```

A.3 Python Testing Code

The following Python codes consist of a code which tests all the sequences and a conversion code. The raw results are saved to "pickle" files, then converted to ".mat" files to be analyzed in a separate MATLAB code.

A.3.1 Overall Test Code

The original overall test code is supplied by Alex Gutzler and can be found in [6]. The code has been modified for this thesis.

```
import sys
import numpy as np
import os
import random
import time
import matplotlib.pyplot as plt
import scipy.special as spc
import pickle as p
import NIST_Suite_Updatedcopy as nist # This is the Python code containing all
#of the NIST tests; it must be included in the same folder as this file
```

```
np.set_printoptions(threshold=sys.maxsize)

# Dictionary of Tests
tests = {'testlist': [nist.monobitfrequencytest,
    nist.blockfrequencytest,
    nist.runstest,
    nist.spectraltest,
    nist.nonoverlappingtemplatematchingtest,
    nist.overlappingtemplatematchingtest,
    nist.serialtest,
    nist.cumultativesumstest,
    nist.aproximateentropytest,
    nist.randomexcursionsvarianttest,
    nist.linearcomplexitytest,
    nist.maurersuniversalstatistictest,
    nist.randomexcursionstest,
    nist.binarymatrixranktest,
    nist.longestrunones10000]}


rlist = ['Mono', 'Block', 'Runs', 'Spectral', 'Non-Overlap', 'Overlap',
        'Serial', 'CumSum', 'Approx Entroy', 'Random Ex Var', 'Linear Complex',
        'Universal', 'Random Ex', 'Rank', 'Longest Runs']

results = {str(item): np.array([]) for item in rlist}

# Where to grab file(s) to be tested from.
cwd = os.getcwd()
rd = os.path.join(cwd,'FolderName\\') #Replace FolderName with actual name of
# folder containing all of the sequences (.txt files)
namelist = os.listdir(rd)
namelist = namelist[0:100]
bl = 10**6 #length in bits to be tested

# Read each file into an array, remove unwanted characters, then run the tests
```

```
for item in namelist:

    remove_from_strings = [" ", "[", "]", ",", "\\n", ".","'","\n"]
    readst = ['1','0',"'","[","]"]
    t0 = time.time()
    inputdata = open(rd+item,'r')
    x = inputdata.readlines()
    inputdata.close()
    x = str(x)
    data = x
#    # Remove unwanted characters
    for s in remove_from_strings:
        data = data.replace(s, "")

    # Run tests
    data = data[0:bl]
    s = 0
    for i in range(0,len(data)):
        s = s+int(data[i]) #Determine the number of 1's in the sequence
    print('Testing commenced')
    for i in range(0,len(rlist)):
        if s < 20000: #If the number of 1's in the sequence is less than 2%,
            #then the sequence will fail all tests
            if i == 6:
                results[rlist[i]] = np.append(results[rlist[i]],np.zeros(2))
            elif i == 9:
                results[rlist[i]] = np.append(results[rlist[i]],np.zeros(18))
            elif i == 12:
                results[rlist[i]] = np.append(results[rlist[i]],np.zeros(8))
            else:
                results[rlist[i]] = np.append(results[rlist[i]],0)
            continue
#The image method caused errors for the longest run of ones test when varying
#th1 and th2. The next 2 lines are required to bypass those errors and result
#in failure for that particular test.
```

```
#        elif i == 14:
#            results[rlist[i]] = np.append(results[rlist[i]],0)
        else:
            results[rlist[i]] = np.append(results[rlist[i]],tests['testlist']\
                  [i](data))
    print(item)
    print('Test Complete')
    print("--- %.5s seconds ---" % (time.time()-t0))
# Average results across all trials
avgresults = {str(item): np.array([]) for item in rlist}
numpass = {str(item): np.array([]) for item in rlist}
for i in range(0,len(rlist)):
    if i == 6:
        avgresults[rlist[i]] = np.append(avgresults[rlist[i]],
                  np.mean(results[rlist[i]].reshape(-1, 2), axis=0))
        numpass[rlist[i]] = np.append(numpass[rlist[i]],
                np.sum(results[rlist[i]].reshape(-1, 2)>.01,axis=0))
    elif i == 9:
        avgresults[rlist[i]] = np.append(avgresults[rlist[i]],
                  np.mean(results[rlist[i]].reshape(-1, 18), axis=0))
        numpass[rlist[i]] = np.append(numpass[rlist[i]],
                np.sum(results[rlist[i]].reshape(-1, 18)>.01,axis=0))
    elif i == 12:
        avgresults[rlist[i]] = np.append(avgresults[rlist[i]],
                  np.mean(results[rlist[i]].reshape(-1, 8), axis=0))
        numpass[rlist[i]] = np.append(numpass[rlist[i]],
                np.sum(results[rlist[i]].reshape(-1, 8)>.01,axis=0))
    else:
        avgresults[rlist[i]] = np.append(avgresults[rlist[i]],
                  np.mean(results[rlist[i]]))
        numpass[rlist[i]] = np.append(numpass[rlist[i]],sum(results[rlist[i]]\
                >.01))
with open('ith2-avgresults.pickle','wb') as f:
    p.dump(avgresults,f)
with open('ith2-results.pickle','wb') as f:
```

```
    p.dump(results,f)
with open('ith2-numpass.pickle','wb') as f:
    p.dump(numpass,f)
```

A.3.2 NIST Test Code

The NIST test code can be downloaded from [4]. However, the original code is designed for Python 2.66 and must be formatted for Python 3.7 to work properly.

A.3.3 Conversion Code

```
import pickle as p
import numpy
import scipy.io

namelist = ['eth1', 'eth2'] #list of file prefixes of files to be converted
for i in namelist:
    ar = p.load(open(i+'-avgresults.pickle','rb'))
    r = p.load(open(i+'-results.pickle','rb'))
    n = p.load(open(i+'-numpass.pickle','rb'))
    scipy.io.savemat(i+'-avgresults.mat',mdict = {'ar':ar})
    scipy.io.savemat(i+'-results.mat',mdict = {'r':r})
    scipy.io.savemat(i+'-numpass.mat',mdict = {'np':ar})
```

A.4 MATLAB Analysis Code

The following MATLAB code uses the results obtained from the NIST tests to determine the pass proportions as well as the p-value distributions.

```
clear
clc
close all
na = {'eth1' 'eth2'}; %list of file prefixes of files to be analyzed
sub = {'-results.mat'};
ci = [.99-3*sqrt(.99*.01/100) 1]; %confidence interval
```

```
t = 0:16;
cu = ones(1,17); cl = ci(1)*ones(1,17); g = .99*ones(1,17);
k=1;
for i = 1:length(na)
    res{k} = load([na{i} sub{1}]);
    k = k+1;
end
k = 1;
fn = fieldnames(res{1}.r);
%The following loop determines the percentage of sequences passing each
%test. Since some tests included multiple parts, the divisor is greater
%than 100 for those cases.
for i = 1:length(na)
    for j = 1:length(fn)
        if j == 7
            np(j,k) = sum(res{i}.r.(fn{j})>=0.01)/200;
        elseif j == 10
            np(j,k) = sum(res{i}.r.(fn{j})>=0.01)/1800;
        elseif j == 13
            np(j,k) = sum(res{i}.r.(fn{j})>=0.01)/800;
        else
            np(j,k) = sum(res{i}.r.(fn{j})>=0.01)/100;
        end
    end
    k = k+1;
end
k = 1;
%Generate figure for pass proportions
for i = 1:length(na)
    figure(k)
    plot(1:15,np(:,i),'k.','MarkerSize',10)
    hold on
    plot(t,cl,'r',t,cu,'r',t,g,'g')
    xlim([1 15]); xlabel('Test'); ylabel('Pass Proportion'); xticks(1:15);
    xticklabels(fn); xtickangle(45);
```

```
    k = k+1;
end
%Generate figure for p-value distribution
for i = 1:length(na)
    figure(k)
    rr = cell2mat(struct2cell(res{i}.r)');
    histogram(rr,10,'BinWidth',0.1,'BinLimits',[0 1])
    xlim([0 1]); xlabel('P-Value'); ylabel('Counts');
    k=k+1;
end
```

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] T. Shinbrot, C. Grebogi, J. Wisdom, and J. A. Yorke, "Chaos in a double pendulum," *American Journal of Physics*, vol. 60, no. 6, pp. 491–499, 1992.

[2] R. Senkerik, M. Pluhacek, and Z. Kominkova-Oplatkova, "Simulation of time-continuous chaotic systems for the generating of random numbers," in *Proceedings of the 18th International Conference on Systems (part of CSCC 2014), Santorini Island, Greece*, 2014, pp. 17–21.

[3] L. E. Bassham, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks, and et al., "Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications," Gaithersburg, MD, USA, Tech. Rep., 2010.

[4] I. Gerhardt, "Random Number Testing," Accessed: Jan 11 2020. [Online]. Available: https://gerhardt.ch/random.php

[5] J. M. Sachs, "Galois field GF2 arithmetic for Python." Accessed Feb 25, 2020. [Online]. Available: https://bitbucket.org/jason_s/libgf2/src/default/src/libgf2/util.py

[6] A. Gutzler, "Chaotic combiner for linear feedback shift register sequences," M.S. Thesis, Dept. Applied Math., Naval Postgraduate School, Monterey, CA, U.S., 2020.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California