12-2013

Application of Linear Sequences to Cryptography

Amanda C. Yeates

The University of Southern Mississippi

Application of Linear Sequences to Cryptography

by

Amanda Corey Yeates

A Thesis
Submitted to the Honors College of
The University of Southern Mississippi
in Partial Fulfillment
of the Requirements for the Degree of
Bachelor of Science
in the Department of Mathematics

December 2013

Approved by

James V. Lambers

Department of Mathematics

The University of Southern Mississippi

Samuel J. Lyle

Department of Mathematics

The University of Southern Mississippi

Sungwook Lee

Department of Mathematics, Chair

The University of Southern Mississippi

David R. Davies

Honors College, Dean

The University of Southern Mississippi

Abstract

Cryptography is the study of a centuries–old technique of secretly transferring information between parties. Linear recurrences were the chosen method of encryption and decryption in the thesis. The Fibonacci sequence, with its Zeckendorf representation, allows for the flexibility of encoding any number desired based on a particular encoding technique used in the film *Sherlock Holmes: A Game of Shadows*. The main goal is to find other linear recurrences that possess characteristics similar to the Fibonacci sequence to use as suitable substitutes for encoding.

Different sequences were analyzed based on a number of criteria. In order for a sequence to be a candidate, it had to be first deemed a possible sequence based on the nature of the roots of its characteristic equation. Once it passed this test, a particular method was developed for showing that a sequence could be used to encode a set of numbers. This method was applied to various sequences, showing which sequences satisfy the desired encryption method.

Key Words: complete sequences, Fibonacci sequence, sequence-based cryptography

Table of Contents

Chapter 1

Introduction

In the movie *Sherlock Holmes: A Game of Shadows*, Professor Moriarty attempts, and at first succeeds, in deceiving Sherlock Holmes about where he will strike his next target. Rather than being obvious by whispering to his evil henchmen the location of the next assassination, he chooses to put the locations in a special numeric code delivered between the members of his group. This idea is actually very old, an art known as cryptography, which traces back several hundred years. Cryptography (a branch of mathematics) is the study of methods to transfer information to another party securely without it being uncovered and plans being intercepted (Luciano & Prichett, 1987). However, Holmes ultimately figures out that his messages are in code, learns the location of his next target by deciphering it, and foils Moriarty's plan just like in the classic hero-villain tale.

While information can be encoded using all sorts of methods involving letters, matrices, and even symbols, the method of encryption here is sequence-based cryptography. The code that Moriarty uses is based on the (mathematically) well-known Pascal's triangle and Fibonacci numbers, the sums of the first diagonals of the triangle. The Fibonacci numbers are written consecutively so that each is a sum of the previous two numbers; this can be represented by the equation $F_p(n) = F_p(n-1) + F_p(n-p-1)$, where $p = 1$ and $F_p(0) = 1$ and $F_p(1) = 1$. The versatility of the previous equation comes from the flexibility of the value of p; by letting p be a different value for each set of encryptions, it makes the code more difficult to decipher.

After choosing to use his horticultural book as a cipher, he formulates a basic code through which he and his men can easily encode and decode messages to each other so that they can communicate covertly. Moriarty first decides on his message then constructs it by choosing letters from his book. He then encodes these page, line, and character numbers

using his encryption method. With their own copy of the book, his colleagues simply take the encoded numbers from the message and, using a particular starting value with the correct sequence, decipher which letters the numbers represent, revealing the entire message (Goriely & Moulton, 2012). While anyone could read the string of numbers, they could not decipher the numbers without knowledge of the decryption process, the correct book for a cipher, or starting value of the sequence. This is an example of how cryptography plays such a crucial role; it allows the passage of important information that seems highly unprotected while remaining indecipherable.

Some codes that are meant to be kept private only work with one sequence of numbers, or key. This key must be used in every encryption and decryption, or the decryption will result in a message that consists of a string of letters making little to no sense. In this case, the same key is used to both encode and decode a sequence. If the incorrect key were to be used with a sequence of numbers meant for one key, then the result would not be correct. It would be like trying to insert an actual key into a vehicle: the key may or may not be able to jam into the car, but it still will not crank the vehicle if it is not an exact match.

When a sequence is to be used as a key, it must consistently serve as the encoder and decoder. Potential sequences must satisfy the following property of being complete, that every positive integer can be represented as a distinct sum of terms in the sequence (Brown, 1961). One goal is to construct an algorithm to indicate sequences that may serve as keys, and to determine methods for identifying sequences that cannot serve as keys. The Fibonacci sequence is a possible key because each positive number can be represented as a sum of terms in the sequence (Zeckendorf, 1972). Other sequences, however, may have problems where certain numbers cannot be expressed as a sum of terms, and therefore cannot be encoded. Hence, the main goal is to identify other sequences of numbers similar to the Fibonacci sequence that can serve as a functional key; the Tribonacci sequence is a

good candidate. It is similar to the Fibonacci sequence yet has different properties because it takes a sum of three numbers to make the next term as opposed to two. Other linear recursive sequences may also be possible candidates.

Chapter 2

Literature Review

Whenever one party needs to communicate information to another without sharing it with outside sources, there needs to be a way to pass the information on securely without it being intercepted or damaged. A safe way of doing this is encoding the message via cryptography; this is a category of mathematics that involves systematically encoding one message into another. Not all cryptography is done the same way, however. Messages can be encrypted using linear ciphers with numbers or letters, substitution ciphers, polyalphabetic ciphers, etc., just to name a few (Luciano & Prichett, 1987).

In sequence-based cryptography, all numbers that are to be encrypted must be successfully translated into another set of values. The Fibonacci sequence works with the code represented in the movie because each integer n can be represented by a sum of non-consecutive Fibonacci numbers. In fact, each number n can be represented uniquely as a sum of distinct non-consecutive sequence terms (Zeckendorf, 1972). This representation (a "Zeckendorf representation") was first made public by Lekkerkerker (1952); however, Edouard Zeckendorf formulated his proof of the representation much earlier in 1939, even though it was not published until later (Kimberling, 1998). Therefore, each positive integer n that corresponds to a particular page, line, or character number that Moriarty would have chosen to encode is able to be encrypted without any problems. No matter which integer is to be encoded, some other numbers of the sequence do sum to it and can be chosen as the set of numbers that replaces that page, line, or character number. Then, this number is translated into the specific term number of the sequence, which forms the coded numbers that Moriarty actually sends in his message (Goriely & Moulton, 2012). In addition, as each n also has a unique representation in terms of the Fibonacci numbers, one number would not have the same coded value as another number. This helps eliminate any confusion in the decryption process as well, since no two values can correspond to the same

plaintext message.

Because the idea of cryptography is to transmit information under the radar, the sequence to be used should be well disguised. The Fibonacci sequence is such a widely known mathematical sequence that Sherlock Holmes was able to easily decipher Moriarty's code and foil his plans. Being able to utilize different sequences within the same code makes the code more versatile and harder to crack. However, as mentioned earlier, each number must be able to be encoded. Using the sequence of the set of even integers for encryption quickly falters because an odd number cannot be written as a sum of even integers. An example of this is that an odd-numbered page could not be encoded into a sum of any even terms because it is impossible to generate that odd number from only even numbers. According to Daykin (1960), no other sequence of numbers besides the Fibonacci sequence exhibits the property of having a Zeckendorf representation, that is, that any integer n can be expressed as a unique sum of non-consecutive sequence elements (Zeckendorf, 1972). A sequence that is complete (Brown, 1961) can work without the restraint of using only unique or non-consecutive sequence elements as with a Zeckendorf representation. Furthermore, depending on the nature of the numbers that are trying to be encoded, a specific sequence could serve as a suitable cipher with limited exceptions, such as choosing a lower limit for the integer to be encoded.

In order to find another sequence to fit Moriarty's encryption style, the code to be used should have similar properties as the Fibonacci sequence. The Tribonacci sequence may be one such sequence. The Tribonacci numbers are defined as $T_n = T_{n-3} + T_{n-2} + T_{n-1}$ where three numbers sum to the next; this is a very similar sequence to the Fibonacci numbers (Feinberg, 1963). The difference lies in that the three previous terms sum up to the next term rather than the two previous terms. These numbers can be generated from a triangle just like the Fibonacci numbers can (Edwards, 2008/2009). Because of the similarity between the Tribonacci numbers and the Fibonacci numbers, it may be possible

to form an altered Zeckendorf representation for these numbers.

Chapter 3

Methodology

While Moriarty uses the popular Fibonacci sequence, other linear sequences have similar patterns. Therefore, one question to ask is what other sequences can serve as a proper key for this particular code. Testing hundreds or thousands of integers in a sequence can become very tedious if done by hand, but a computer can perform these calculations in fractions of a second. Therefore, any needed computations will be written in a computer program known as Sage (Stein, 2012) to allow them to be done in mass quantity in an ideal amount of time.

In order to test sequences for the code, the sequences must first be created. In order to do so, this will require writing a code in Sage that can generate a desired number of terms of a sequence based on the recurrence for the sequence. For example, when the first 3 integers of the equation of the Tribonacci sequence are given, it can compute the next, say 100 integers of the sequence, based on the equation $T_n = T_{n-1} + T_{n-2} + T_{n-3}$, which explains that the next term of the sequence is given by the sum of previous three terms. This code can be altered to generate a sequence with any number of starting terms and any linear recurrence. A linear recurrence is a sequence in which the next number in the sequence is created by adding linear combinations of previous terms of the sequence with initial values established. For example, the Fibonacci sequence, $F_n = F_{n-1} + F_{n-2}$, is one such that F_n is formed by the linear combinations of coefficients (equal to 1) multiplied by the previous two consecutive terms. Because of how Sage works, any recursive sequence can be generated using this approach.

The idea of cryptography is to write one number as another; therefore, it must be possible to write each positive integer as a sum of the numbers of the sequence being used for the encryption. For a sequence to be acceptable for use as a key, it must be complete

(Brown, 1961). After generating a sequence, there must be a way to test this "rewriting process" with any integer. One method to obtain representations that may work is the use of a greedy algorithm, where a representation is the form of writing a single integer n as a sum of elements from a particular sequence. This is like a change-making algorithm, and it involves taking the largest term of a sequence that goes into the integer n then subtracting it from n leaving a smaller value. Unless the difference is 0, the process continues to repeat until the difference of 0 is reached, resulting in a sum of terms from the sequence that makes up n. Again, because this can be tedious, writing a function in Sage will allow this greedy algorithm to be run on any given integer for a particular sequence. This can allow the testing of hundreds of integers, which can eliminate or suggest sequences as possibilities for keys of the code.

Chapter 4

Complete Sequences

4.1 Fibonacci Numbers and the Code

As already described, each letter of the message refers to a specific letter of a specific line of a specific page. The page, line, and character numbers have to then be converted into the coded numbers. Before they can be converted, a key must be formulated as a basis for the conversion. The Fibonacci sequence is used as the key for encoding and decoding the message in the movie *Sherlock Holmes: A Game of Shadows*.

The Fibonacci sequence is a linear recurrence of the form $F_n = F_{n-1} + F_{n-2}$; thus, each term is made by summing the previous two terms of the sequence. Altering the definition slightly can lead to an entire family of recurrences based on the Fibonacci sequence. A value p can be used in the sequence to give an altered form of the following:

$$F_p(j) = F_p(j-1) + F_p(j-p-1).$$

For the Fibonacci sequence, this p-value is 1, resulting in the sequence $1,2,3,5,8,13,\ldots$ (or sometimes $1,1,2,3,5,8,13,\ldots$) . To add variation to the sequence, the value of p can simply be changed so that the sequence will follow a different pattern. For encryption purposes, choosing a p-value other than 1 may decrease the likelihood that the sequence will be known, thus heightening the security of the encryption.

In order to begin the encryption of a number, it is crucial to know the initial p-value. Each number n is encoded as a codeword consisting of the sequence indices of the sequence terms that add to achieve n. The process begins with choosing the largest term less than or equal to n and proceeds by subtracting that largest term from n. Putting together the values that sum to n in this manner is a concept known as the greedy algorithm. The indices of

the terms are then used to encode that number. For example, using $p = 3$ (with the indices beginning at 0 rather than 1, which is the sequence used in Goriely & Moulton, 2012):

Sequence: $0, 1, 2, 3, 4, 5, 7, 10, 14, 19, 26, 36, 50, 59, 85, 121, \ldots$

23 = 19 + 4, encoded as 0409
72 = 59 + 10 + 3, encoded as 030713
100 = 85 + 14 + 1, encoded as 010814

In the above example, the integer 23 is found by first subtracting the largest number from the sequence that is less than or equal to 23. After 19 is subtracted, the remaining 4 is also a term of the sequence. The index of 4 is 04, and the index of 19 is 09, using two digits for consistency. Thus, the encoded form of 23 is 0409. The following numbers are encoded in the same fashion.

To decode a set of numbers, the same concept applies in reverse. The main necessity for the decoding party is the initial p-value so the decoder knows what sequence he/she is working with. After acquiring that, the sequence can be found from the equation, then the encoded numbers are simply the term numbers. Those terms can be added up to equal the final value, which stands for a specific page, line, or character number. This method of encryption and decryption is adapted from the method used in *Sherlock Holmes* (Goriely & Moulton, 2012).

4.2 Zeckendorf Representation and Brown's Theorem

In the previous example that was used to encode a set of numbers, the greedy algorithm was used to find which terms of the sequence added to make the number of choice. In order for this to be possible, particular terms of the sequence must be available in order to make those sums. The Fibonacci sequence possesses the virtue that every positive integer can be expressed as a unique sum of non-consecutive Fibonacci numbers (Zeckendorf,

1972).

The basic concept behind finding the Zeckendorf representation for the Fibonacci sequence is using the greedy algorithm. An integer n is chosen for which to find the Fibonacci numbers that sum to it. An observation made in the proof by Zeckendorf is that no two consecutive Fibonacci numbers are included in the sum representation, as two consecutive numbers are equivalent to the next Fibonacci number in the sequence. It has already been stated that each Zeckendorf representation is unique, that is an integer n can only have one representation. An example of a sequence that does not have a unique representation is $1,2,4,6,8,10,12,14,16,18,20,22,\ldots$. Using the greedy algorithm, the number 10 could be written as a sum of the sequence elements $10 = 8+2$, but it could also be written as $10 = 6+4$. Both representations allow the number 10 to be written as a sum of sequence elements, but the second one uses consecutive sequence terms.

In order to make other sequences work, it is necessary to be able to rewrite integers as sums of their sequence elements just as the Fibonacci sequence. It has been shown that no other linear recurrence has the capability of representing integers as sums of unique, non-consecutive sequence elements (Daykin, 1960). However, unique representations as a sum of non-consecutive sequence terms is not necessary. All that is necessary for encryption and decryption is that the sequence is complete; neither consecutive terms nor non-uniqueness have any detrimental effect on encryption or decryption.

Brown (1961) showed that, given a nondecreasing sequence of positive integers with $f_1 = 1$, the sequence $\{f_i\}$ is complete if and only if $f_{p+1} \leq 1+\sum_1^p f_i$ for $p = 1,2,\ldots$. Also in this paper is a corollary to this theorem stating that if a sequence consists of nondecreasing positive integers, then any element, say f_i, of that sequence will be less than or equal to 2^{i-1}. In Chapter 5, it will be shown that a sequence can have the possibility of being complete only if the roots of the characteristic equation of the sequence satisfy $-2 < r_i < 2$. The closed form of a sequence can be found via two methods, either generating functions or

the characteristic equation of a sequence. The Fibonacci sequence, having a Zeckendorf representation and roots $\frac{1+\sqrt{5}}{2}$ and $\frac{1-\sqrt{5}}{2}$, is also a complete sequence. The closed form of the Fibonacci sequence can be written as

$$F_j = -0.447213595499958\left(\frac{1-\sqrt{5}}{2}\right)^j + 0.447213595499958\left(\frac{1+\sqrt{5}}{2}\right)^j$$

where the closed form coefficients of the sequence have been computed. But it is possible for a sequence to be complete without having a Zeckendorf representation because the sum does not have to consist of non-consecutive sequence elements. It is necessary to require distinct terms from the sequence in a representation of an integer.

It is advantageous to utilize the property of completeness to analyze other sequences for their possibility as candidates for encryption. If the roots of the characteristic equation of a sequence are outside the accepted interval, then the sequence can quickly be eliminated. Once this test has been performed on a sequence and passed, other methods can be used to further analyze if integers can be encoded using terms of the sequence.

Chapter 5

Linear Recurrences

5.1 Generating Functions

Based on a previously mentioned corollary of Brown's theorem, one method to determine if a sequence is not complete is by going through numerous elements of a sequence until a sequence term fails $f_i \leq 2^{i-1}$. However, that can become time-consuming or may not occur for a long time, if at all. One way to go about quickly identifying sequences that are not complete is through the use of generating functions. (For a more extensive look at generating functions, see Tucker, 2012.)

The formula for a particular generating function can be expressed as $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + \cdots$, where a_i are the terms of the sequence. Using this equation, the following displays an example for how a generating function can be used on the Fibonacci sequence, which is of the form $a_n = a_{n-1} + a_{n-2}$, to obtain a closed form of the sequence for analysis of its roots.

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + \cdots \quad (1)$$

$$xf(x) = \quad a_0x + a_1x^2 + a_2x^3 + a_3x^4 + a_4x^5 + \cdots \quad (2)$$

$$x^2f(x) = \quad a_0x^2 + a_1x^3 + a_2x^4 + a_3x^5 + \cdots \quad (3)$$

Line 1 represents the basic equation for a generating function for the sequence. In Line 2, the equation is being multiplied by x because the term a_n is created by first shifting back one term before adding it to its previous term. So multiplying by x in a sense "shifts" the equation back a term. Likewise, in Line 3, the equation is multiplied by x^2 because this causes the sequence to shift back another term. The terms that not all three equations have in common are being set equal to an arbitrary j. Part of the first line of the next equation,

$f(x) = xf(x) + x^2 f(x)$, is derived from the formula of the sequence $a_n = a_{n-1} + a_{n-2}$.

$$\begin{aligned} f(x) &= xf(x) + x^2 f(x) + j \\ f(x) - xf(x) - x^2 f(x) &= j \\ f(x)(1 - x - x^2) &= j \\ f(x) &= \frac{j}{1 - x - x^2} \\ f(x) &= \frac{-j}{x^2 + x - 1} \end{aligned}$$

Next, the denominator of $f(x)$ can be factored by the quadratic equation.

$$x = \frac{-1 \pm \sqrt{(1)^2 - 4(1)(-1)}}{2(1)} = \frac{-1 \pm \sqrt{5}}{2}$$

The roots can be set up as if the equation were going to be solved via partial fractions.

$$\frac{-j}{x^2 + x - 1} = \frac{A}{x - \left(\frac{-1+\sqrt{5}}{2}\right)} + \frac{B}{x - \left(\frac{-1-\sqrt{5}}{2}\right)}$$

The denominator of each partial fraction can be rearranged so that the root is in the geometric series form $(1 - ax)$.

$$\text{Positive Root :} \qquad x - \left(\frac{-1+\sqrt{5}}{2}\right) = \frac{1-\sqrt{5}}{2}\left(1 - \frac{2}{-1+\sqrt{5}}x\right)$$

$$\text{Negative Root :} \qquad x - \left(\frac{-1-\sqrt{5}}{2}\right) = \frac{1+\sqrt{5}}{2}\left(1 - \frac{2}{-1-\sqrt{5}}x\right)$$

Each root can be used to write the closed form of the sequence.

$$a_n = C_1\left(\frac{2}{-1-\sqrt{5}}\right)^n + C_2\left(\frac{2}{-1+\sqrt{5}}\right)^n$$

Note that, when rationalized, $\frac{2}{-1-\sqrt{5}} = \frac{1-\sqrt{5}}{2}$ and $\frac{2}{-1+\sqrt{5}} = \frac{1+\sqrt{5}}{2}$.

When the larger root is evaluated, $\left(\frac{2}{-1+\sqrt{5}}\right) \approx 1.6$. Since $1.6 < 2$, 1.6^n increases at a slower rate than 2^n. This information infers that the sequence has a possibility of being complete. (The Fibonacci sequence has already been proved to be complete, but this is an example of how the approach of using generating functions can render possible complete sequences.) When the smaller root is evaluated, $\left(\frac{2}{-1-\sqrt{5}}\right) \approx -0.6$, which eventually approaches zero when raised to the nth power. Therefore, the previous equation can be written as the following:

$$a_n \approx C_2 \left(\frac{2}{-1+\sqrt{5}}\right)^n.$$

Another example for using a generating function to check the possible completeness on a sequence will be shown below. The sequence of interest is a linear sequence, but different from the previous example, as the coefficients of the previous two terms are no longer 1. A linear sequence is one such that each term of the sequence can be found via an equation using the previous terms of the sequence. The sequence to be used is $a_n = 4a_{n-1} + 3a_{n-2}$.

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + \cdots \tag{4}$$

$$4xf(x) = \quad 4a_0x + 4a_1x^2 + 4a_2x^3 + 4a_3x^4 + 4a_4x^5 + \cdots \tag{5}$$

$$3x^2f(x) = \quad 3a_0x^2 + 3a_1x^3 + 3a_2x^4 + 3a_3x^5 + \cdots \tag{6}$$

The same manipulations to the generating function are being applied just as with the previous example. Whenever a coefficient is present, that number is also multiplied by each term in the function in addition to power of x being multiplied by the function. Again, the

value of j will represent the difference between the three equations, in terms of recurrence.

$$\begin{aligned} f(x) &= 4xf(x) + 3x^2 f(x) + j \\ f(x) - 4xf(x) - 3x^2 f(x) &= j \\ f(x)(1 - 4x - 3x^2) &= j \\ f(x) &= \frac{j}{1 - 4x - 3x^2} \\ f(x) &= \frac{-j}{3x^2 + 4x - 1} \end{aligned}$$

After shifting the original generating function, the differences between the equations are being set equal to $f(x)$. The equation can then be solved for $f(x)$, where the denominator is then factored by the quadratic equation.

$$x = \frac{-4 \pm \sqrt{(4)^2 - 4(3)(-1)}}{2(3)} = \frac{-2 \pm \sqrt{7}}{3}$$

The roots are then set up into partial fractions, so as to see that they can then be rearranged into a form of the geometric series, $(1 - ax)$.

$$\frac{-j}{3x^2 + 4x - 1} = \frac{A}{x - \left(\frac{-2+\sqrt{7}}{3}\right)} + \frac{B}{x - \left(\frac{-2-\sqrt{7}}{3}\right)}$$

$$\text{Positive Root :} \qquad x - \left(\frac{-2+\sqrt{7}}{3}\right) = -\frac{-2+\sqrt{7}}{3}\left(1 - \frac{3}{-2+\sqrt{7}}x\right)$$

$$\text{Negative Root :} \qquad x - \left(\frac{-2-\sqrt{7}}{3}\right) = -\frac{-2-\sqrt{7}}{3}\left(1 - \frac{3}{-2-\sqrt{7}}x\right)$$

The following is the closed form of the sequence in terms of the roots and unknown closed form coefficients.

$$a_n = C_1\left(\frac{3}{-2-\sqrt{7}}\right)^n + C_2\left(\frac{3}{-2+\sqrt{7}}\right)^n$$

When evaluated, the smaller root $\left(\frac{3}{-2-\sqrt{7}}\right) \approx -1.5$. Depending on the parity of n, this

number raised to the nth power may have a positive or negative contribution to the sequence. The larger root, $\left(\frac{3}{-2+\sqrt{7}}\right) \approx 4.6 > 2$, and therefore 4.6^n increases at a faster rate than 2^n. Because one of the roots dissatisfies the root conditions, this sequence does not satisfy the conditions of being complete, and therefore no further testing is necessary.

5.2 Characteristic Equations

The characteristic equation is a useful tool that can be used to help check the completeness of a sequence by allowing for the determination of the closed form of a sequence. It works in a manner analogous to generating functions, except the method is much quicker. (For a complete description of characteristic equations, see Tucker, 2012.)

When given the formula for the terms of a sequence, the first step is to rearrange the equation so that it is equal to zero. From there, the coefficients can be extracted from the terms of the sequence and paired with corresponding x-values. For example, using the formula for the Fibonacci sequence:

$$\begin{aligned} a_n &= a_{n-1} + a_{n-2}, \\ a_n - a_{n-1} - a_{n-2} &= 0, \\ x^2 - x - 1 &= 0. \end{aligned}$$

Assuming that $a_n = x^n$, the last line of the equation is translated from the previous. Then $a_{n-1} = x^{n-1}$, and $a_{n-2} = x^{n-2}$. Because the Fibonacci sequence is a two term recurrence, $n = 2$, yielding $a_n = x^2$, $a_{n-1} = x$, and $a_{n-2} = 1$.

Using the equation now at hand, the roots of the equation can now be computed. Using the same example as above, the following roots are found for the Fibonacci sequence

using the quadratic formula:

$$x = \frac{1 \pm \sqrt{(-1)^2 - 4(1)(-1)}}{2(1)},$$
$$x = \frac{1 \pm \sqrt{5}}{2}.$$

Now that the roots of the equation have been found, this leads to being able to determine if the sequence may or may not be complete. If the absolute values of the roots are all less than or equal to 2, then the sequence has a possibility of being complete. On the other hand, if any of the absolute values of the roots are greater than 2, this implies that the sequence will not be complete. However, the coefficients of the roots of the sequence in the closed form must be taken into account. If a root has an absolute value that is greater than or equal to 2, but its coefficient is zero, then the root has no contribution to the sequence. Therefore this root can be ignored when checking for completeness of the sequence. When generating a formula for the terms of the sequence using the roots, the coefficients can cause certain roots to be ignored. The formula for finding the terms of the sequence is as follows, given that the roots are distinct (where r_i are the roots of the sequence and c_i are the closed form coefficients):

$$a_k = c_1 r_1^k + c_2 r_2^k + \cdots + c_n r_n^k$$

If the roots repeat, the equation changes only slightly. For the following example, the root r_1 is repeated three times. Then the first time r_1 appears in the closed form of the sequence, it is paired with the coefficient c_1. To account for the first repetition of r_1, a factor of k is introduced and the product is multiplied by c_2. The second repetition of r_1 consists of a factor of k^2, with the next coefficient c_3. The remaining roots are written as linear

combinations as usual:

$$a_k = c_1 r_1^k + c_2 k r_1^k + c_3 k^2 r_1^k + c_4 r_4^k + \cdots + c_n r_n^k.$$

If several of the coefficients c_i were to be zero, then the sequence would have a possibility of being complete with the given initial conditions regardless of the roots. When testing for the completeness of the sequence, it is necessary to see that the coefficients of roots with absolute values greater than 2 not have coefficients of zero. Because the terms of the sequence are known in addition to the roots, the coefficients can be found. The number of unknown coefficients is equivalent to the value of k of the desired term a_k. This results in a set of equations numbered 1 to k. Since there are an equal number of unknown coefficients as there are equations to solve for, then a system of equations can be used to solve for the coefficients. The following function was written and tested in Sage to test both the roots and coefficients, allowing the user to input any number of initial values and coefficients for the linear sequence. (Details of the exact Sage code can be found in the Appendix.)

```
complete([1,1],[1,1])
    (
    [-0.447213595499958]
    [ 0.447213595499958], ''This sequence may be complete.''
    )
```

Based on the results from the computed function, the chosen initial values and coefficients are those from a sequence that meets the necessary condition on the roots of the characteristic equation (for completeness). A different example, such as the one shown below, can

be ruled out immediately based on its initial values and coefficients.

$$a_n = 3a_{n-1} + 4a_{n-2}$$

$$a_n - 3a_{n-1} - 4a_{n-2} = 0$$

$$x^2 - 3x - 4 = 0$$

$$x = \frac{-4 \pm \sqrt{(4)^2 - 4(3)(-1)}}{2(3)} = \frac{-2 \pm \sqrt{7}}{3}$$

```
complete([1,1],[3,4])
    ''This sequence cannot be complete.''
```

Without even running the initial values and coefficients of the sequence through the code, speculation of the roots shows that the sequence will not be complete simply because one of the roots is larger than 2, while the other is negligible. However, the Sage function analyzes the closed form coefficients that are multiplied by the roots of the sequence to see that roots with absolute values larger than 2 have nonzero coefficients. The function only considers roots of the characteristic equation with nonzero coefficients.

5.3 Generating Sequences

In order to work with a linear sequence, such as rewriting an integer as a sum of the sequence elements, it is first necessary to generate the elements of the sequence. The Fibonacci sequence, for example, is created by summing the previous two terms to get the next term. It is quite tedious to manually calculate many terms of a sequence, and many sequences for that matter. This type of process can be quickly accomplished by a computer, therefore different functions can be written to find the number of terms of a sequence, a particular term from a list of terms, or access which term number from a sequence an element is.

The following function is a short function that is an example of one that would be used to find a given number of terms for the Fibonacci sequence.

```
def Fibonacci_sequence(n):
    terms = []
    terms.append(1)
    terms.append(1)
    for i in range(2,n+1):
        T = terms[i-1] + terms[i-1]
        terms.append[T]
    return terms
```

The user would simply choose a number of terms n and the code would compute n terms of the Fibonacci sequence in seconds. The above code illustrates the general mechanism that can be used to generate sequences that have any chosen starting values and any chosen coefficients. The Appendix contains the details of the function that is used to find the terms of the sequence given particular initial values and coefficients.

Chapter 6

Step-by-Step Process of Proving a Sequence is Complete

6.1 A Modification of Brown's Theorem

Brown's theorem shows that a sequence of nondecreasing positive integers with $f_1 = 1$ is complete if it satisfies $f_{n+1} \leq 1 + \sum_1^n f_i$ for $n = 1, 2, \ldots$ (Brown, 1961). We want to show that the same result holds if $f_{n+1} \leq 1 + \sum_1^n f_i$ for $n \geq N_0$, and that $m \geq f_{N_0}$ can be represented as a sum of sequence elements.

Theorem 1. *Suppose f_i is a nondecreasing positive sequence with the property that every $m < 1 + \sum_{i=1}^{N_0} f_i$ can be represented as a sum of distinct sequence terms (which avoid f_{N_0+1}). Then $f_{n+1} \leq 1 + \sum_{i=1}^{n} f_i$ for $n > N_0$ implies $\{f_i\}$ is complete.*

Proof. Suppose that, for $m \leq M_0$ where $M_0 = 1 + \sum_{i=1}^{N_0} f_i$, m can be written as a sum of distinct sequence elements such that for the smallest m, f_{m+1} (the next element in the sequence) is not used. It is necessary to show that

$$f_{n+1} \leq 1 + \sum_{i=1}^{n} f_i$$

for $n \leq N_0$ for $\{f_i\}_{i=1}^{\infty}$, implies $\{f_i\}$ is complete. This modifies Brown's theorem by relaxing the condition that $N_0 = 1$. To show this, consider m such that

$$1 + \sum_{i=1}^{n} f_i < m \leq 1 + \sum_{i=1}^{n+1} f_i$$

where $n \geq N_0$. The values of m such that $m \leq 1 + \sum_{i=1}^{N_0} f_i$ are handled either by assumption or induction. That is, it can be assumed that the integers less than m can be expressed as sums of sequence elements. Then, by the condition that $f_{N+1} \leq 1 + \sum_{i+1}^{n} f_i$ (since $n \geq N_0$), $m - f_{n+1} > 0$. Also, since $m \leq 1 + \sum_{i+1}^{n+1} f_i$, then $m - f_{n+1} \leq 1 + \sum_{i=1}^{n} f_i$ by transitivity. Then, $m - f_{n+1}$ can be represented as a sum. The term f_{n+1} can then be added to $m - f_{n+1}$

to obtain m. (It can be assumed that for $m \leq 1 + \sum_{i=1}^{n} f_i$, m can be written as a sum of distinct sequence elements such that f_{n+1} is not used.) $\square$

6.2 Overview of the Process

For the scope of this project, results were obtained for sequences that satisfy the following properties:

1. All roots are real,
2. all coefficients are nonzero,
3. and there are no repeated roots.

Suppose that

$$f_n = c_1 r_1^n + c_2 r_2^n + \cdots + c_k r_k^n$$

The goal here is to show that for $n > N_0$

$$\begin{aligned} f_{n+1} &\leq 1 + \sum_{l=1}^{n} f_l \\ &= 1 + \sum_{l=1}^{n} c_1 r_1^l + c_2 r_2^l + \cdots + c_k r_k^l \\ &= 1 + c_1 \left(\sum_{l=1}^{n} r_1^l \right) + c_2 \left(\sum_{l=1}^{n} r_2^l \right) + \cdots + c_k \left(\sum_{l=1}^{n} r_k^l \right). \end{aligned}$$

It can be assumed that $|r_i| < 2$ or $c_i = 0$, as a sequence will have been tested by Sage functions up to this point.

Let a be a computed root of the equation, and assume that $a \neq 1$. The previous sum $\sum_{l=1}^{n} r_k^l$ can be written as a geometric series as follows: $a + a^2 + a^3 + a^4 + \cdots + a^n$. The following manipulations can be done on the series to get the series to the form $\frac{a^{n+1}-a}{a-1}$. Let

S simply be the sum of the series. Then

$$S = a + a^2 + a^3 + a^4 + \cdots + a^n$$
$$aS = a^2 + a^3 + a^4 + \cdots + a^n + a^{n+1}$$

Subtracting aS from S,

$$\begin{aligned} S - aS &= (a + a^2 + a^3 + a^4 + \cdots + a^n) - (a^2 + a^3 + a^4 + \cdots + a^n + a^{n+1}) \\ S(1-a) &= a - a^{n+1} \\ S &= \frac{a - a^{n+1}}{1-a} \\ S &= \frac{a^{n+1} - a}{a-1} \end{aligned}$$

Now the sums of the powers of the roots can be rewritten in the form $\frac{a^{n+1}-a}{a-1}$.

$$\begin{aligned} f_{n+1} &\leq 1 + \sum_1^n f_l \\ &= 1 + c_1 \left(\frac{r_1^{n+1} - r_1}{r_1 - 1} \right) + c_2 \left(\frac{r_2^{n+1} - r_2}{r_2 - 1} \right) + \cdots + c_k \left(\frac{r_k^{n+r} - r_k}{r_k - 1} \right) \end{aligned}$$

Now that each geometric series has been replaced by a closed form, a sequence of steps can be taken to check if a particular sequence is complete. For a linear recurrence, it is first necessary to find the closed form of the sequence. That is, to find each of the roots and their corresponding coefficients that form a linear combination for the next term of the sequence. Unless the sequence is ruled as incomplete, then this process progresses from here.

Using the closed form for $1 + \sum_1^n f_i$ (for which each term above was determined to be

of the form $\frac{a^{n+1}-a}{a-1}$), each root can be compared individually, solving for n for the following:

$$r_i^{n+1} \leq \left(\frac{r_i^{n+1} - r_i}{r_i - 1} \right).$$

Solving for n for each root will result in a set of different values of n. These values can then be compared, where the largest will serve as a lower bound for which integers can be represented as sequence elements.

Ideally, the main objective is to be able to represent any integer m as a sum of the elements of the sequence of interest, which makes it complete. To verify that our sequence is complete starting from a specific integer $m = 1 + \sum_{i=1}^{N_0} f_i$, the functions in Sage can be performed on the initial values and coefficients of the sequence.

Using the extension of Brown's theorem, the next step is to be able to determine N_0 for Brown's Lemma so that if f_i is increasing, then $f_{n+1} \leq 1 + \sum_{i=1}^{n} f_i$ for $n \geq N_0$. Taking that inequality, it can be manipulated into another way of writing the inequality to solve for n:

$$c_1 r_1^{n+1} + c_2 r_2^{n+1} \leq 1 + c_1 \left(\frac{r_1^{n+1} - r_1}{r_1 - 1} \right) + c_2 \left(\frac{r_2^{n+1} - r_2}{r_2 - 1} \right)$$

While Sage can be used to solve for n once the roots of the characteristic equation have been computed, the following shows patterns of how the value of n will behave depending on which interval it lies within $(-2, 2)$.

6.3 Analyzing Roots

The goal is to obtain a lower bound for n, and the work below shows for which roots this can be accomplished.

Condition: $r_i = 0$

$$0^{n+1} \leq (0^1 + 0^2 + 0^3 + \cdots + 0^n)$$
$$0 \leq 0$$

Since the above case is always true, there is no restriction on n.

Condition: $r_i = 1$

$$1^{n+1} \leq (1^1 + 1^2 + 1^3 + \cdots + 1^n)$$
$$1 \leq n$$
$$n \geq 1$$

Sequences typically begin at 1, therefore the n will always be at least 1. Now since the above case is always true, there is no restriction on n.

Condition: $r_i = -1$

$$-1^{n+1} \leq (-1^1 + -1^2 + -1^3 + \cdots + 1^n)$$
$$1 \leq -1 \quad \text{(result if } n \text{ is odd)}$$
$$-1 \leq 0 \quad \text{(result if } n \text{ is even)}$$

If the value of n is odd, then the result of the inequality is a false expression of $1 \leq -1$. However, if the value of n is even, the result is a true inequality. A root of -1 thus has a parity restriction on solving for n.

Condition: $1 < r < 2$

$$
\begin{aligned}
r_i^{n+1} &\leq \frac{r_i^{n+1} - r_i}{r_i - 1} \\
r_i^{n+1}(r_i - 1) &\leq r_i^{n+1} - r_i \\
r_i^{n+2} - r_i^{n+1} &\leq r_i^{n+1} - r_i \\
r_i^{n+2} - 2r_i^{n+1} &\leq -r_i \\
-r_i^{n+2} + 2r_i^{n+1} &\geq r_i \\
r_i^{n+1}(2 - r_i) &\geq r_i \\
2 - r_i &\geq \frac{r_i}{r_i^{n+1}} \\
2 - r_i &\geq r_i^{-n} \\
\underbrace{\ln(2 - r_i)}_{\text{negative}} &\geq -n \underbrace{\ln(r_i)}_{\text{positive}} \\
\frac{\ln(2 - r_i)}{\ln(r_i)} &\geq -n \\
-\frac{\ln(2 - r_i)}{\ln(r_i)} &\leq n \\
n &\geq \frac{-\ln(2 - r_i)}{\ln(r_i)}
\end{aligned}
$$

Here there are two sign changes (signified by two changes in color), resulting in a lower bound on n.

Condition: $0 < r_i < 1$

$$r_i^{n+1} \leq \underbrace{\frac{r_i^{n+1} - r_i}{r_i - 1}}_{\text{negative}}$$

$$r_i^{n+1}(r_i - 1) \geq r_i^{n+1} - r_i$$

$$r_i^{n+2} - r_i^{n+1} \geq r_i^{n+1} - r_i$$

$$r_i^{n+2} - 2r_i^{n+1} \geq -r_i$$

$$-r_i^{n+2} + 2r_i^{n+1} \leq r_i$$

$$r_i^{n+1}(2 - r_i) \leq r_i$$

$$2 - r_i \leq \frac{r_i}{r_i^{n+1}}$$

$$2 - r_i \leq r_i^{-n}$$

$$\ln\left(\underbrace{2 - r_i}_{\text{positive}}\right) \leq -n \underbrace{\ln(r_i)}_{\text{negative}}$$

$$\underbrace{\frac{\ln(2 - r_i)}{\ln(r_i)}}_{\text{negative}} \geq -n$$

$$\frac{-\ln(2 - r_i)}{\ln(r_i)} \leq n$$

$$n \geq \frac{-\ln(2 - r_i)}{\ln(r_i)}$$

Here there are four sign changes (signified by four changes in color), resulting in a lower bound for n.

From the above algebraic manipulations, it is shown that for any roots larger than 0 (and of course below 2 for the sequence to be complete), that the value of n should be greater than or equal to $\frac{-\ln(2-r_i)}{\ln(r_i)}$. The next cases that will be handled will be for negative roots.

Condition: $-1 < r_i < 0$

$$r_i^{n+1} \leq \frac{r_i^{n+1} - r_i}{\underbrace{r_i - 1}_{\text{negative}}}$$

$$r_i^{n+1}(r_i - 1) \geq r_i^{n+1} - r_i$$

$$r_i^{n+2} - r_i^{n+1} \geq r_i^{n+1} - r_i$$

$$r_i^{n+2} - 2r_i^{n+1} \geq -r_i$$

$$-r_i^{n+2} + 2r_i^{n+1} \leq r_i$$

$$r_i^{n+1}(2 - r_i) \leq r_i$$

Here, we have two different cases for how n will play out depending on if n is odd or even. Below we show the outcome of the two.

If n is odd, then r_i^{n+1} is positive.

$$r_i^{n+1}(2 - r_i) \leq r_i$$

$$2 - r_i \leq \frac{r_i}{r_i^{n+1}}$$

$$\underbrace{2 - r_i}_{\text{positive}} \leq \underbrace{r_i^{-n}}_{\text{negative}}$$

If n is even, then r_i^{n+1} is negative.

$$r_i^{n+1}(2 - r_i) \leq r_i$$

$$2 - r_i \geq \frac{r_i}{r_i^{n+1}}$$

$$\underbrace{2 - r_i}_{\text{positive}} \geq \underbrace{r_i^{-n}}_{\text{positive}}$$

$$2 - r_i \geq (-1)^{-n}(r_i)^{-n}$$

$$2 - r_i \geq (1)(r_i)^{-n}$$

$$2 - r_i \geq (-r_i)^{-n}$$

$$\ln(2 - r_i) \geq -n\ln(-r_i)$$

$$\frac{\ln(2 - r_i)}{\ln(-r_i)} \leq -n$$

$$-\frac{\ln(2 - r_i)}{\ln(-r_i)} \geq n$$

The above situation when n is odd yields a positive solution being greater than a negative solution, but this is not possible. However, when n is even the solution is possible with the value of n begin smaller than some value.

Condition: $-2 < r_i < -1$

$$r_i^{n+1} \leq \frac{r_i^{n+1} - r_i}{\underbrace{r_i - 1}_{\text{negative}}}$$

$$r_i^{n+1}(r_i - 1) \geq r_i^{n+1} - r_i$$

$$r_i^{n+2} - r_i^{n+1} \geq r_i^{n+1} - r_i$$

$$r_i^{n+2} - 2r_i^{n+1} \geq -r_i$$

$$-r_i^{n+2} + 2r_i^{n+1} \leq r_i$$

$$r_i^{n+1}(2 - r_i) \leq r_i$$

As with the previous condition, there are again two different cases for how n will play out depending on if n is odd or even. Below is the outcome of the two different parity options.

If n is odd, then r_i^{n+1} is positive.

$$r_i^{n+1}(2 - r_i) \leq r_i$$

$$2 - r_i \leq \frac{r_i}{r_i^{n+1}}$$

$$\underbrace{2 - r_i}_{\text{positive}} \leq \underbrace{r_i^{-n}}_{\text{negative}}$$

If n is even, then r_i^{n+1} is negative.

$$r_i^{n+1}(2 - r_i) \leq r_i$$

$$2 - r_i \geq \frac{r_i}{r_i^{n+1}}$$

$$\underbrace{2 - r_i}_{\text{positive}} \geq \underbrace{r_i^{-n}}_{\text{positive}}$$

$$2 - r_i \geq (-1)^{-n}(r_i)^{-n}$$

$$2 - r_i \geq (1)(r_i)^{-n}$$

$$2 - r_i \geq (-r_i)^{-n}$$

$$\ln(2-r_i) \geq -n\ln(-r_i)$$
$$\frac{\ln(2-r_i)}{\ln(-r_i)} \geq -n$$
$$-\frac{\ln(2-r_i)}{\ln(-r_i)} \leq n$$

The result regarding the parity of n for this condition is the same as before. Whenever there is a negative root, including analyzing the case of -1, n can never be odd. If n is even and lies on the interval $(-2,-1)$, then it will be greater than or equal to some value. However if n is even and lies on the interval $(-1,0)$, n will be less than or equal to some value. With the case of positive roots, the value of n will always be greater than some value with no parity restrictions.

Chapter 7

Application

After going through the steps of determining how a sequence can be deemed complete, a particular sequence other than the Fibonacci sequence can be chosen with which to encode the original page, line, and character numbers, as was the main goal. To begin, the sequence that is chosen will be tested for completeness in Sage to determine whether or not it will be ruled out immediately based upon the values of its roots. The sequence of interest is

$$a_n = 2a_{n-1} - a_{n-3}.$$

The above sequence coefficients are 2, 0, and -1. In order to be complete, the sequence needs to be able to represent 1, so we choose $a_0 = 1$. To finish defining the sequence, we arbitrarily choose $a_1 = 2$ and $a_2 = 4$. From there the sequence was run through the following function in order to obtain the possibility of completeness and yielded the following results.

```
complete([1,2,4],[2,0,-1])
    ([-0.170820393249937]
    [ -1.00000000000000]
    [  1.17082039324994], ‘‘This sequence may be complete.’’)
```

The closed form coefficients have now been determined to be -0.1708, -1.000, and 1.1708, in addition to the possibility of the sequence being complete. Now the next step is to determine the roots of the sequence. The following function utilizing the sequence coefficients allowed for the computation of the roots of the sequence.

```
characteristic([2,0,-1])
    [-0.618033988749895, 1.00000000000000, 1.61803398874989]
```

The roots of the characteristic equation have now been found to be -0.618, 1.000, and 1.618. Combining both the roots and the closed form coefficients, each subsequent term of the sequence can be written as

$$a_n = (-0.1708)(-0.618)^n + (-1.000)(1.000)^n + (1.1708)(1.618)^n.$$

Going back to the closed form of $1 + \sum_1^n a_i$, the previously derived inequality that characterizes n is

$$r_i^{n+1} \leq \left(\frac{r_i^{n+1} - r_i}{r_i - 1} \right).$$

The n value for each case for the appropriate value of r_i has already been obtained. The appropriate case will then be chosen so that the values of n can be computed and compared.

```
n1 = -log(2-(1-sqrt(5))/2)/log(-(1-sqrt(5))/2)
n1.n()
    2.00000000000000
n2 = -log(2-(1+sqrt(5))/2)/log((1+sqrt(5))/2)
n2.n()
    2.00000000000000
```

The value for n for the root 1.000 does not have to be computed because there is no restriction on 1 when solving for n. Since the largest computed value of n resulted in $n \geq 2$, it is safe to assume that the sequence is complete starting from the integer satisfying $m = 1 + \sum_{i=0}^{1} a_i$, that is, $m = 4$. (The starting value of the sum has been adjusted because the indexing of the sequence begins at zero, therefore the upper bound must be set at $n-1$; this makes $n - 1 = 1$.) The first four numbers up to 4 can be manually represented by sequence elements using the greedy algorithm.

1 = 1
2 = 2
3 = 2 + 1

4 = 4

The next step would be to encode a particular set of numbers, such as a set of page, line, and character numbers. Let the arbitrarily chosen values be page 329, line 23, and character 45. The greedy algorithm function in Sage is then used to determine which numbers of the sequence make up the values to be encoded.

```
greedy([1,2,4],[2,0,-1], 329)
    ([232, 88, 7, 2], 4)
greedy([1,2,4],[2,0,-1], 23)
    ([20, 2, 1], 3)
greedy([1,2,4],[2,0,-1], 45)
    ([33, 12], 2)
```

After determining which terms of the sequence will be used to encode the original page, line, and character numbers, the indices of the terms must be recorded. Taking the largest sequence term necessary to encode a number, each term number can be found using the following function in Sage.

```
S = sequence([1,2,4],[2,0,-1], 12)
S
    [1, 2, 4, 7, 12, 20, 33, 54, 88, 143, 232, 376]
A = 1 + S.index(232)
A
    11
```

The last step would repeat until all term numbers are found. The following page, line, and character numbers would be written as follows:

Page 329 = 2 + 7 + 88 + 232 = 02040911
Line 23 = 1 + 2 + 20 = 010206

Character 45 = 12 + 33 = 0507

Therefore page 329, line 23, character 45 would be encoded as

02040911 010206 0507

which is an example of encoding a single letter on a page.

Chapter 8

Conclusion

Because every integer can be written as a sum of its sequence elements, the Fibonacci sequence is obviously one of the most useful linear recurrences that can be used for cryptographic applications. Exhibiting the properties of completeness and having a Zeckendorf representation, it is a rank above all other linear sequences. However, it is also a model, with its many properties, which other sequences will be compared to.

Through the use of generating functions, the characteristic equation, and computer programming, it is possible to analyze various parts of a linear sequence. Sequence coefficients, initial values, closed form coefficients, and the roots of a sequence all play vital roles in the behavior of a sequence from how it grows to if it will be able to be used for encoding numbers.

Even though the Fibonacci sequence seems like the perfect candidate for every occasion, the original idea behind cryptography is to be as secretive as possible. The only danger with the Fibonacci sequence is that it is highly recognizable. Being able to find other sequences that fit the bill for encryption of this sort decreases the chances for deciphering, which is the ultimate goal when passing secret messages. As shown in the application of using an alternative sequence, Sherlock Holmes may not have as easily uncovered the locations of escapade if a different sequence had been chosen. In order to encode (and decode) a message using an alternative linear recurrence, much analysis and thought must go into the process of choosing a sequence that is suitable, from the terms of the sequence all the way down to its roots.

References

Brown, Jr., J. L. (1961). Note on Complete Sequences of Integers. *The American Mathematical Monthly, 68, 6*, pp. 557-560.

Daykin, D. E. (1960). Representation of Natural Numbers as Sums of Generalised Fibonacci Numbers. *Journal of the London Mathematical Society, s1-35, 2*, 143-160.

Edwards, K. (2008/2009). A Pascal-like triangle related to the Tribonacci numbers. *The Fibonacci Quarterly, Volume 46/47, no. 1*, pp. 18-25.

Feinberg, M. (1963). Fibonacci-Tribonacci. *The Fibonacci Quarterly, Volume 1, 3*. pp. 71-74.

Goriely, A., & Moulton, D. E. (2012 April). The Mathematics Behind *Sherlock Holmes: A Game of Shadows*. *SIAM News, 45*, 1.

Kimberling, C. (1998). Edouard Zeckendorf. *The Fibonacci Quarterly, Volume 36*, pp. 416-418.

Lekkerkerker, C. G. (1952). Voorstelling van natuurlijke getallen door een som van getallen van Fibonacci. *Simon Stevin, Volume 29*, pp. 190-195.

Leonardo of Pisa (1202). *Liber Abaci*.

Luciano, D. & Prichett, G. (1987). Cryptology: from Caesar Ciphers to Public-Key Cryptosystems. *The College Mathematics Journal, Volume 18, 1*, pp. 2-17.

Tucker, A. (2012). *Applied Combinatorics*. New York: John Wiley & Sons, Inc.

Zeckendorf, E. (1972). Représentation des nombres naturels par une somme de nombres de Fibonacci ou de nombres de Lucas. *Bull. Soc. R. Sci. Liége, 41*, pp. 79-182.

Appendix

Generating Sequences

The following function can be used to find any given number of terms for the Fibonacci sequence. Following the function are two examples of how the function finds *n* terms of the Fibonacci sequence.

```
def Fibonacci_sequence(n):
#input is the number of terms desired
    terms = []
    terms.append(0)
    terms.append(1)
    for i in range(2,n+1):
        T = terms[i-2] + terms[i-1]
        terms.append(T)
    return terms
#output the generated terms of the sequence


Fibonacci_sequence(9)
    [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
Fibonacci_sequence(14)
    [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
```

The next function is analogous to the previous function, except that it finds any given number of terms for a Tribonacci sequence. This type of sequence takes into account three initial values, so that the fourth term (and every term thereafter) is found by adding the previous three. Following the function are two examples of how the function finds *n* terms of a Tribonacci sequence.

```
def tribonacci_sequence(a1,a2,a3,n):
#inputs are the three initial values and the desired
#number of terms
    terms = []
    terms.append(a1)
    terms.append(a2)
    terms.append(a3)
    for i in range(3,n+1):
        T = terms[i-3] + terms[i-2] + terms[i-1]
        terms.append(T)
    return terms
#output the generated terms of the sequence


tribonacci_sequence(1,2,3,11)
        [1, 2, 3, 6, 11, 20, 37, 68, 125, 230, 423, 778]
tribonacci_sequence(2,3,5,9)
        [2, 3, 5, 10, 18, 33, 61, 112, 206, 379]
```

Not all sequences follow the pattern of the Fibonacci or Tribonacci sequences, so another function must be written to formulate sequences with any given initial coefficients and any coefficients. The following function displays a function that does so, with two examples to follow.

```
def sequence(IV,CF,k):
#input a list of initial values, list of coefficients
#for the sequence, and the desired number of terms
    LIV = len(IV)
    LCF = len(CF)
```

```
    RCF = CF
    RCF.reverse()
    if LIV != LCF:
        print ``This sequence will not work!''
    else:
        for j in range(LIV,k):
            T = 0
            for i in range(LCF):
                T = T + RCF[i]*IV[i+j-LIV]
            IV.append(T)
        return IV
#output the generated terms of the sequence

sequence([1,1],[1,1],15)
    [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610]
sequence([1,2],[3,4],8)
    [1, 2, 10, 38, 154, 614, 2458, 9830]
```

Complete Sequences

This first function computes the roots of a sequence by the process of finding the characteristic equation using the sequence coefficients. It is followed by two examples of how a sequence's roots are computed.

```
def characteristic(CF):
#input a list of sequence coefficients
    LCF = len(CF)
    x = PolynomialRing(CC,'x').gen()
    f = x^LCF
```

```
    for i in range(LCF):
        f = f - CF[i]*x^(LCF - i - 1)
    roots = f.roots()
    L = len(roots)
    R = []
    for i in range(L):
        RP = roots[i]
        for j in range(RP[1]):
            R.append(RP[0])
    return R
#output the roots of the sequence


characteristic([2,2])
    [-0.732050807568877, 2.73205080756888]
characteristic([1,1,1])
    [1.83928675521416, -0.419643377607081 -
    0.606290729207199*I, - 0.419643377607081 +
    0.606290729207199*I]
```

This function returns whether or not the sequence may or may not be complete, given the particular initial values and sequence coefficients. It finds the roots of the function (using a previous function) and determines whether they fall in the range $-2 < r_i < 2$. After finding these roots, it determines the closed form coefficients to determine whether a sequence can actually be ruled as not complete or possibly complete. There are also two examples of sequences being tested for completeness.

```
def complete(IV,CF):
#input a list of initial values and list of coefficients
```

```
#of the sequence
    LIV = len(IV)
    LCF = len(CF)
    if LIV != LCF:
        print ``Try again.''
        return
    R = characteristic(CF)
    RootMatrix = matrix(CC,LCF,LCF)
    k = 0
    for j in range(LCF):
        if (j > 0) and (R[j] == R[j-1]):
            k = k + 1
        else:
            k = 0
        for i in range(LCF):
            RootMatrix[i,j] = (R[j]^(i+1))*(i+1)^k
    SequenceMatrix = matrix(CC,LCF,1)
    for i in range(LCF):
        SequenceMatrix[i] = IV[i]
    CoefficientMatrix = RootMatrix\SequenceMatrix
    for i in range(LCF):
        if ((R[i] > 2 or R[i] < -2) and (CoefficientMatrix[i]
        != 0)):
            return ``This sequence cannot be complete.''
    return CoefficientMatrix, ``This sequence may be
    complete.''
#output if the sequence may complete or not
```

```
#if the sequence is complete, also include the closed
#form coefficients


complete([1,1],[1,1])
    (
    [-0.447213595499958]
    [ 0.447213595499958], ``This sequence may be complete.''
    )
complete([1,1],[3,4])
    ``This sequence cannot be complete.''
```

Using the Greedy Algorithm

Since the idea behind the Zeckendorf representation is to use the greedy algorithm, the following function employs that particular algorithm on the Fibonacci sequence, followed by two examples of an integer being broken down by it, including how many terms it takes to make up the integer. It consists of the basic mechanism for how the greedy algorithm works.

```
def greedy_fib(m,n):
#input the integer to be encoded and the number of sequence
#terms to encode the integer using a previous function
    F = Fibonacci_sequence(n)
    F.reverse()
    S = []
    while m > 0:
        while F[0] > m:
            F.remove(F[0])
        m = m - F[0]
```

```
        S.append(F[0])
        F.remove(F[0])
    L = len(S)
    return S, L
#output the terms that sum to the integer and the
#number of terms needed for the sum

greedy_fib(213,20)
    ([144, 55, 13, 1], 4)
greedy_fib(1281,15)
    ([987, 233, 55, 5, 1], 5)
```

The next function is analogous to the function that determines if a sequence is complete. It only determines the closed form coefficients for the sequence, which can be used for the greedy algorithm.

```
def coefficients(IV,CF,R):
#input a list of initial values, a list of coefficients,
#and a list of roots of a sequence
    LIV = len(IV)
    LCF = len(CF)
    if LIV != LCF:
        print ``Try again.''
        return
    RootMatrix = matrix(CC,LCF,LCF)
    k = 0
    for j in range(LCF):
        if (j > 0) and (R[j] == R[j-1]):
```

```
            k = k + 1
        else:
            k = 0
        for i in range(LCF):
            RootMatrix[i,j] = (R[j]^(i+1))*(i+1)^k
    SequenceMatrix = matrix(CC,LCF,1)
    for i in range(LCF):
        SequenceMatrix[i] = IV[i]
    CoefficientMatrix = RootMatrix\SequenceMatrix
    return CoefficientMatrix
#output the closed form coefficients of the sequence

coefficients([1,1],[1,1],characteristic([1,1]))
    [-0.447213595499958]
    [ 0.447213595499958]
```

The final function employs the greedy algorithm on any sequence, using any given initial values and coefficients.

```
def greedy(IV,CF,m):
#input a list of initial values and a list of coefficients
#of the sequence and an integer to be encoded
    R = characteristic(CF)
    CM = coefficients(IV,CF,R)
    RL = len(R)
    C = []
    D = []
    for i in range(RL):
```

```
        if abs(R[i]) > 1:
            C.append(abs(R[i]))
            D.append(CM[i,0])
    CL = len(C)
    minimum = 3
    for j in range(CL):
        if C[j] <= minimum:
            minimum = C[j]
                location = j
    p = minimum
    c = abs(D[location])
    n = log(m/c,p)
    q = n.n()
    n = q.ceil()
    S = sequence(IV,CF,n)
    S.reverse()
    G = []
    while m > 0:
        while S[0] > m:
            S.remove(S[0])
        m = m - S[0]
        G.append(S[0])
        S.remove(S[0])
    L = len(G)
    return G, L
#output the terms necessary to sum to the integer m and
#the number of terms it takes
```

```
greedy([1,2,3],[1,1,1],564)
    ([423, 125, 11, 3, 2], 5)
greedy([1,2,4],[2,0,-1],123987)
    ([121392, 2583, 12], 3)
```