Texas A&M University

Research Project 1

Some Conjectures on Fibonacci Numbers

Authors:
Daniel Whatley
Sarah Sahibzada
Taylor Wilson

Supervisor:
Dr. Sara Pollock

November 4, 2015

Contents

1 Introduction

The *Fibonacci sequence*, $\{F_n\}$, is defined as follows:

$$F_1 = 1, \qquad F_2 = 1, \qquad F_n = F_{n-1} + F_{n-2} \quad \text{for} \quad n \geq 3,$$

with seed values $F_0 = 0$ and $F_1 = 1$. Similarly, the Lucas numbers are a sequence of integers defined by the following recurrence:

$$L_n = L_{n-1} + L_{n-2}$$

with seed values $L_1 = 1$ and $L_2 = 3$; alternative definitions begin this recurrence as $L_0 = 2$ and $L_1 = 1$. In this paper we analyze two conjectures: one about which Fibonacci numbers are perfect squares, and on various identities linking the Fibonacci sequence and the Lucas numbers.

These numbers are known to be closely related to the Fibonacci sequence. This was investigated computationally.

A number of identities were verified computationally up to the first 175 Lucas and Fibonacci numbers as well.

2 Theoretical Analysis

2.1 Fibonacci Squares

It was a long-standing conjecture, until it was proved in 1964 [2], that the only Fibonacci numbers that are perfect squares are $F_1 = 1$, $F_2 = 1$, and $F_{12} = 144$. M. Wunderlich[1] described an ingenious "exclusion" method to calculate which Fibonacci numbers, out of the first million, can possibly be perfect squares. The method does not prove that any Fibonacci numbers are perfect squares, rather it rules out those that are not.

In Section 3, we describe some limitations of this approach, and describe an optimization.

2.2 Lucas Numbers and Associated Identities

Insofar as the Fibonacci and Lucas sequences, and their sums, may be developed in different ways with various identities and recurrences, a large number of identities connecting these two related sequences are known. These identities serve as the computational basis for many conjectures and family of sums involving the Lucas and Fibonacci sequences, as proposed in "Sums of Certain Products of Fibonacci and Lucas Numbers".

$F_{n+k} + F_{n-k} = F_n L_k$, even k
$F_{n+k} + F_{n-k} = L_n F_k$, odd k
$F_{n+k} - F_{n-k} = F_n L_k$, odd k
$F_{n+k} - F_{n-k} = L_n F_k$, even k
$L_{n+k} + L_{n-k} = L_n L_k$, even k
$L_{n+k} - L_{n-k} = F5_n F_k$, even k
$L_{n+k} + L_{n-k} = F_n L_k$, even k
$L_n^2 - L_{2n} = -2 = -L_0$, odd n
$5F_{2n}^2 - L_{2n}^2 = -4 = -L_0^2$
$5F_{2n}^2 - L_{4n} = -2 = L_0$

It is possible to obtain individual Lucas numbers from Fibonacci numbers using various identities relating these quantities:

$L_n = F_{n-1} + F_{n+1}$

$$\begin{bmatrix} L_{n+1} \\ L_n \end{bmatrix} = Q_L \begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix}$$

as well as

$$5 \begin{bmatrix} F_{n+1} \\ L_n \end{bmatrix} = Q_L \begin{bmatrix} L_n \\ L_{n-1} \end{bmatrix}$$

, where we define $Q_L = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}$

The identities relating the Fibonacci and Lucas numbers are significant in their use proving the following theorems for the construction of a family of sums of the Fibonacci numbers. *Theorem 1:*

Theorem 2.1. $\Sigma_{k=1}^{n} F_k F_{k+1} ... F_{k+4m+2} Lk + 2m + 1 = \frac{F_n Fn+1 ... Fn+4m+3}{F_{2m+2}}$

,

Theorem 2.2. $\Sigma_{k=1}^{n} L_k L_{k+1} ... L_{k+4m+2} F_{k+2m+1} = [\frac{L_k L_{k+1} ... L_{k+4m+3}}{5F_{2m+2}}]_0^n$

Theorem 2:

Theorem 2.3. $\Sigma_{k=1}^{n} F_k^2 F_{k+1}^2 ... F_{k+4m}^2 F_{2k+4m} = \frac{F_n^2 F_{n+1}^2 ... F_{n+4m+1}}{F_{4m+2}}$

,

Theorem 2.4. $\Sigma_{k=1}^{n} L_k^2 L_{k+1}^2 ... L_{k+4m}^2 F_{2k+4m} = [\frac{L_k^2 L_{k+1}^2 ... L_{k+4m+1}^2}{5F_{4m+2}}]_0^n$

3 Computational Approach

3.1 Implementation Detail

Various languages and operating systems were used to implement each part of this investigation. Particularly, the Fibonacci and Lucas generators were implemented in Python 2.7 and run on a Mac operating system, while the code to verify the Fibonacci and Lucas identities was written in Java. Due to the portability of the Java Virtual Machine, the operating systems on which all Java code was run were not kept consistent. Finally, C/C++ code was written and run on both a Mac operating system and Windows 8 in both the Dev-C++ IDE and on the Mac terminal. Also utilized was the Ada supercomputer based in Texas A&M University. Ada has a Linux-based operating system, 845 compute nodes, and 20 cores per node; its peak performance is 337 teraflops, or 10^{12} floating-point operations per second. Libraries used for computations included GNU Multi-Precision Arithmetic Library (GMP) for working with some large numbers in the C++ code on the Fibonacci squares; Sage, an open-source, Python-based mathematical language with support for various number-theoretic functions; and BigInteger, a Java API allowing for arbitrary-precision arithmetic. Parallel code in C++ utilized the POSIX threads, while parallel code in Java utilized the Runnable interface.

3.2 Fibonacci Squares

[1] found that the only Fibonacci numbers up to $F_{1000000}$ are the three described above using a computational approach. The researchers here took a similar approach, but used optimization techniques to decrease running time and to give more conclusive results. The programming language used in this research was predominantly C++.

Consider a prime p. We first calculate the period of the Fibonacci numbers $\pmod p$, denoted here as the *Fibonacci period* of p. For example, the Fibonacci period of $p = 7$ is 16, as the Fibonacci sequence $\pmod 7$ repeats after 16 steps. Specifically, the first 16 Fibonacci numbers $\pmod 7$ are:

$$1, 1, 2, 3, 5, 1, 6, 0, 6, 6, 5, 4, 2, 6, 1, 0,$$

after which it repeats $1, 1, 2, 3, 5, \ldots$ again. Because the numbers $\pmod 7$ that are quadratic non-residues are 3, 5, and 6, replacing each quadratic residue by a 1 and each quadratic non-residue by a 0 gives the binary string

$$111001010001101.$$

Let this string for each prime p be S_p.

Suppose we take a list of primes: $p_1, p_2, \ldots, p_n$, and let the Fibonacci periods of each prime be $d_1, d_2, \ldots, d_n$. Take those indices of S_{p_i} that are 0. If any Fibonacci number is congruent to one of these indices mod d_i, then it can never be a perfect square of any integer. Such Fibonacci numbers can now be eliminated due to this prime. Repeat for more primes, until the number of possible square Fibonacci numbers is down to a reasonable number to analyze. We expect that approximately half of the Fibonacci numbers will be eliminated at each step, as about half of the integers mod a prime p are quadratic residues. If $L = \operatorname{lcm}(d_1, d_2, \ldots, d_n)$, the worst case running time of this algorithm is $O(L \log n)$, where n is the number of primes considered.

The computation in [1] was somewhat limited for two reasons. First, for each prime p, the quadratic residues and non-residues mod p had to be re-iterated all the way to the computation limit (in the paper, the computation limit is 10^6). This means that a loop of size 10^6 had to be performed each time a new prime was considered. The computation here is superior for another reason: any F_k with $k \not\equiv a \pmod{L}$ where F_a is a perfect square is therefore also not a perfect square. This is because if $k \not\equiv a \pmod{L}$, then there exists some d_i for which $k \not\equiv a \pmod{d}_i$, which also means that F_k is not a quadratic residue modulo p_i, which means F_k cannot be a perfect square. Because 10^6 is not perfectly divisible by each of the Fibonacci periods considered, the computation cannot easily be extended to all positive integers, but the computation here can. This is how the approach was optimized.

3.3 Lucas Numbers and Various Identities

4 Results and Conclusion

4.1 Fibonacci Squares

Since the value of L cannot be too large for our purposes (the computational size is linear in L), we must find primes with relatively small Fibonacci periods. To do this, we limited our primes p_i to below 10^5, and gave limits on $v_2(d_i)$, $v_3(d_i)$, $v_5(d_i)$, $v_7(d_i)$, $v_9(d_i)$, $v_{11}(d_i)$ where $v_q(d_i)$ is the power of prime q in the prime factorization of d_i. Different files were generated based on these limits, each specifying a value of L. After this algorithm was run on each file, the possible residues $r_1, r_2, \ldots \pmod{L}$ such that $F_{r_1}, F_{r_2}, \ldots$ can be perfect squares were outputted.

In each run, we got that the only possible Fibonacci numbers that can be perfect squares were the ones with indices $1, 2, 12 \pmod{L}$, confirming the computation done in [1] and the proof done in [2]. Execution times on Ada are summarized as follows:

L	Number of primes	Execution time (s)
604800	103	0.88
907200	112	1.34
6350400	138	9.24
9072000	135	12.37
63504000	163	105.61

4.2 Lucas Numbers and Various Identities

This code was run serially on a Mac operating system; it was not run on Ada. Due to ranging errors, results could be verified only up to $n, k = 250$. Results are below:

Range of n,k	Mac OS (s)	
50	0.274	0.004
100	0.758	0.004
150	1.43	0.004
200	2.30	0.004
250	3.64	0.004

As the means of testing each identity was initially $O(n^2)$, each identity was run in a separate thread of execution. Execution time was recorded on a Mac terminal using the Java Virtual Machine, as well as on Ada. The multi-threaded execution times may be summarized as below:

Range of n,k	Mac OS (s)	Ada (s)
50	0.039	0.004
100	0.059	0.004
150	0.187	0.004
200	0.051	0.004
250	0.094	0.004
300	0.077	0.004
350	0.104	0.005
400	0.189	0.004
450	0.3304	0.003
500	0.094	0.004

The speedup and efficiency up to $n, k = 250$ were calculated, as serial code was available only up to this range and on the Mac OS.

Range of n,k	Speedup	Efficiency
50	0.195	0.001
100	0.084	0.007
150	0.130	0.011
200	0.119	0.010
250	0.025	0.002

Therefore, the maximum speedup obtained over the serial code reduced the execution time to 2.5 percent of its original execution time. No consistent behavior was observed with efficiency, however, except for that the ratio was consistently low. This is likely due to the fact that each computation was performed in an individual thread, thereby utilizing parallelism as a means to accomplish some tasks in parallel, but not taking advantage of the full capacities of either parallelism or the number of threads. It is relevant to note that as the number of threads used in this implementation remained constant, the speedup is simply a scalar multiple of the efficiency, with the other factor being the number of threads used.

Generating Fibonacci and Lucas sequences is a simple task. What is difficult is being able to decide whether a given number belongs to either of the sequences. The general method requires you to start at the beginning and generate the entire list up until the number in question. This is an arduous task and fairly inefficient when working with large numbers. This presents an interesting application to cryptosystems. As mentioned in [5], there are methods of creating keys to crytposystems by using Fibonacci and Lucas sequences to generate a vector. This creates a very secure network because there is only one unique solution to each sequence. By combining our optimized creation of the sequences with the identities computationally verified, we could extend this project into cryptography by attempting to create a unique sequence to be used in a symmetric cryptosystem.

5 Individual Contributions

1. Sarah: Wrote code to verify various identities on Lucas and Fibonacci sums; wrote small scripts to generate Lucas and Fibonacci numbers based on known identities; implemented the Fibonacci squares algorithm in parallel in C; re-implemented the Fibonacci and Lucas sum

code in parallel. Developed some theoretical background on the Lucas sequence. Verified Fibonacci squares conjecture in the supercomputer. Write computational approaches aspect of paper.

2. Daniel: I mainly wrote code for verifying the Fibonacci squares conjecture. I implemented an algorithm to find relatively reasonable Fibonacci periods, generated different files for verification, and implemented various algorithms to find quadratic residues and non-residues, powers of integers modulo others, and other elementary number theory constructs required for thorough analysis of the subject. I also developed theoretical background in this conjecture and for the optimization of the computation in [1]. Finally, I collected results for this conjecture and produced some tables.

3. Taylor: Ran code to verify Lucas conjectures; Provided background research and analysis of results.

6 References

1. M. Wunderlich, On the non-existence of Fibonacci Squares, Maths, of Computation, 17 (1963) p. 455.

2. J. H. E. Cohn, "Square Fibonacci Numbers, Etc." Fibonacci Quarterly **2** 1964, pp. 109-113.

3. Koken, Fikri, and Durmus Bozkurt. "ON LUCAS NUMBERS BY THE MATRIX METHOD." *Hacettepe Journal of Mathematics and Statistics* 39.4 (2010): 471-75. *Hacettepe University Journal of Mathematics and Statistics*. Web. 23 Oct. 2015.

4. Melham, R. S. "SUMS OF CERTAIN PRODUCTS OF FIBONACCI AND LUCAS NUMBERS." *Fibonacci Quarterly* (1999): n. pag. *The Fibonacci Quarterly*. Web. 23 Oct. 2015.

5. Luma, A., and B. Raufi. "Relationship between Fibonacci and Lucas Sequences and Their Application in Symmetric Cryptosystems." *Latest Trends on Circuits, Systems, and Signals* (n.d.): n. pag. *WSEAS Conference*. Web. 23 Oct. 2015.