SIMULATION OF A QUANTUM PRIME FACTORING ALGORITHM

by

Elizabeth E. Parsons

A Thesis Submitted to the Graduate School of

The University of Colorado

in Partial Fulfillment of the Requirements

for the Degree of

Master of Arts

Department of Mathematics

May 2016

Boulder, Colorado

This thesis entitled:

Simulation of a Quantum Prime Factoring Algorithm

written by Elizabeth Ellen Parsons

has been approved for the Department of Mathematics

APPROVED:

Dr. Katherine Stange	Date
Dr. Bengt Fornberg	Date
Dr. Judith Packer	Date
Dr. Divya E. Vernerey	Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

ABSTRACT

Elizabeth Ellen Parsons
MA, Department of Mathematics
Simulation of a Quantum Prime Factoring Algorithm
Advisor: Dr. Katherine Stange

The intent of this thesis is to elucidate the quantum computing algorithm developed by Peter Shor called Shor's algorithm. We will provide a detailed description and simulation of the algorithm using MATLAB. Precursory information regarding quantum phenomena such as superposition, entanglement, and Dirac notation, will be described in great detail so that the reader may have a better understanding of the operations in Shor's algorithm.

Quantum computers store and transport information quite differently than their classical counterparts. We will provide a quick overview of these differences to highlight the benefit of utilizing quantum phenomena in a computer in order to create massive parallel computations. Thus, reducing the complexity time for classical algorithms used to solve problems such as the prime factorization problem and the period finding problem.

The Quantum Fourier Transform is a principle component in Shor's algorithm. We will explicitly define the Quantum Fourier Transform and show that it is a unitary transformation. We will also show how the Quantum Fourier Transform, as well as another unitary transformation called the Hadamard transform, functions in Shor's algorithm.

One of the initial parameters in Shor's algorithm is to select a random variable. We will examine the erratic effects of this random variable as well as how it effects the probability of us successfully reducing an integer into a product of two primes. We will provide a thorough analysis of the randomness in Shor's algorithm. We will also show how measuring the state of our quantum system as well as selecting a suitable random variable impacts finding the period of the Quantum Fourier Transform which in turn will either give a high or low probability of obtaining a factor of some integer.

To my parents,

I am forever grateful for your

unconditional love and support in all that I do.

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude and appreciation for my advisor, Dr. Katherine Stange. I am incredibly grateful for her immeasurable support and encouragement throughout my studies. Her scope of knowledge as well as her patience and flexibility were invaluable. Working with Dr. Stange has been a very rewarding and positive experience. I am deeply indebted to her for devoting so much time and guidance in every aspect of my masters research.

I have been privileged to learn from incredible professors whose enthusiasm for mathematics has been quite influential in my studies. I would like to thank Dr. Bengt Fornberg for inspiring me to explore areas of mathematics which would have gone unnoticed. I am incredibly thankful for his passion of bringing mathematics to life through modeling and simulation. I would also like to thank Dr. Judith Packer for her kindness and ebullience. Her passion for making sure students truly understand mathematics was pivotal for me. Furthermore, I would like to thank Dr. Divya Vernerey. I am exceedingly grateful for her wholehearted compassion and support through difficult times in my studies. Her guidance and understanding were irreplaceable.

Thank you Dr. Bengt Fornberg, Dr. Judith Packer and Dr. Divya Vernerey for your time and willingness to be members of my committee.

Finally, I would like to thank my parents and sister for their unconditional love. I could not have done this without you. Thank you.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

In "Euclidis Elementa", the greek mathematician Euclid proposed that every composite number is measured by some prime number (Heiberg & Fitzpatrick, 2008, p. 218). That is to say, every composite integer can be factored into a product of prime integers. Many modern day cryptosystems utilize this to create secure means of sharing private information. The most commonly used cryptosystem is the asymmetric, also called public-key, cryptographic scheme RSA named after Ronald Rivest, Adi Shamir and Leonard Adleman. The security of RSA relies on the fact that factoring a large composite integer into a product of two prime integers can not be accomplished in polynomial time (Rivest et al., 1977).

We say an algorithm runs in polynomial time, P, if the number of steps required to find the solution for a given input is bounded by a polynomial function $\mathcal{O}(n^k)$ where k is a nonnegative integer and n is the complexity of the input (Terr, 2016). Factoring the product of large prime numbers can be done using sub-exponential algorithms. The run time of sub-exponential algorithms are faster than any exponential algorithm but significantly slower than any polynomial algorithm. Hence, factoring composite integers which are hundreds of digits long into product of two prime integers is impractical to implement on classical computers.

The most efficient classical algorithm used to factor integers is the general number field sieve (GNFS) algorithm. The sub-exponential complexity of the GNFS algorithm is

$$exp\Big(\Big(\frac{64}{9}\Big)^{\frac{1}{3}}(\ln(N)^{\frac{1}{3}})(\ln(\ln(N))^{\frac{2}{3}}\Big).$$

Using the GNFS algorithm, RSA Laboratories estimated factoring a 2048-bit key, an integer 617 digits long, would require 9×10^{15} years to factor on a standard desktop ("RSA Laboratories"). This is more than the number of atoms in the observable universe (4×10^{81} atoms) ("Shannon Number," 2016).

In his paper "Simulating Physics with Computers", Richard Feymann asked: "By using the properties of quantum mechanics in a computer, can we compute more efficiently then on a classical computer?". MIT applied mathematician, Peter Shor, took Feymann's question one step further. He developed an algorithm, now called Shor's Algorithm, which demonstrated that the complexity time it takes to factor a large composite integer into a product of two primes can be reduced to polynomial time if implemented on a quantum computer. Thus, the security of RSA would be compromised since RSA relies solely on the fact that factoring a large integer takes an impractical amount of time to compute.

In 2001, a group at IBM successfully implemented Shor's Algorithm on a quantum computer. They were able to factor 15 into its product of primes 3 and 5 using a Nuclear Magnetic Resonance (NMR) quantum computer with 7-qubits (Vandersypen et al., 2001, p. 883 - 887). Since then, the current largest integer factored on a NMR quantum computer is 56153 (Dattani & Bryans, 2014).

ENTANGLEMENT

Entanglement plays a crucial role in quantum computing. It is thought to be the main reason certain algorithms, especially Shor's Algorithm, can out-perform its classical counterparts (Kendon & Munro, 2005). Quantum computing and quantum information processing utilize the quantum phenomenon of entanglement. Because of entanglement, quantum computers are able to perform arithmetic and logical operations on many qubits simultaneously as compared to a classical computer which can only perform operations in an evolutionary fashion i.e. bit-by-bit.

Quantum entanglement is a physical phenomenon which occurs when pairs or groups of particles behave in such a way that the states of the particles can not be described independently from one another. Altering the state of one particle will effect the state of the other particles. Furthermore, the distance between entangled particles is irrelevant. Meaning, changing the state of one particle will still change the state of the other particles even if they are light years away.

For example, suppose two observers, A and B, are each holding a box with a ball inside. One box contains a red ball and the other box contains a blue ball. Both observers do not know which color of ball they are holding. If observer A opens the box and discovers a red ball, then we know with 100% certainty that observer B must be holding the blue ball. As illustrated in Figure 1, the color of the ball remains unchanged for both observers when the boxes are open and when the boxes are closed.

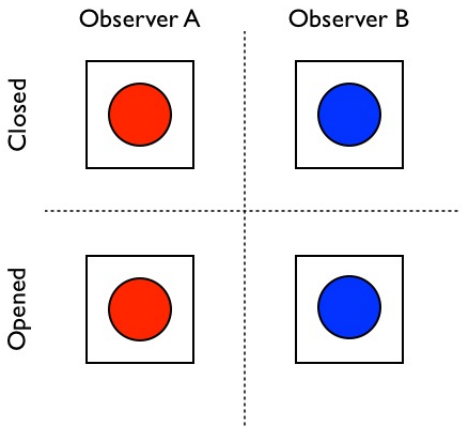


Figure 1: Non-Entangled Balls

Now suppose observer A and observer B are in some parallel universe where each ball can be both red and blue at the same time. We call this superposition. If observer A opens her box and discovers she has a red ball, then we know with 100% certainty observer B is holding a blue ball prior to observer B even opening his box. Remember that observer B also had a ball in superposition, e.g. both red and blue at the same time. Equally important, observer B didn't open his box so how do we know with 100% certainty observer B is holding the blue ball? Something strange happened when observer A opened her box which collapsed the state of observer B's ball. The only way to describe this interaction is to say that observer A and observer B's balls are entangled.

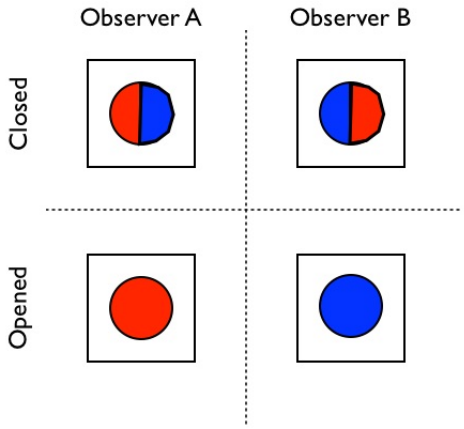


Figure 2: Entangled Balls

DIRAC NOTATION & LINEAR ALGEBRA FOR QUANTUM COMPUTING

Before we represent an entangled quantum state mathematically, we will introduce the Dirac, or Bra-ket, notation. Dirac notation is a different, perhaps more compact way of handling vectors and their corresponding bases. David Mermin described it best. "Mathematicians tend to despise Dirac notation, because it can prevent them from making important distinctions, but physicists love it, because they are always forgetting such distinctions exist and the notation liberates them from having to remember." (Bacon).

In quantum computing, we will almost exclusively be living in a Hilbert space. A Hilbert space, $\mathcal{H}$, is a complete inner product space composed of N-dimensional complex vectors, $\mathbb{C}^N$. Since we will consider only Hilbert spaces which are finite dimensional, we can choose a finite basis of vectors. A basis is a set of linearly

independent vectors, $\{|b_i\rangle : i \geq 0\}$, which span the vector space. Dirac notation denotes column vectors as "kets", $|\psi\rangle$, and their corresponding dual vectors as "bras", $\langle\psi|$, where bras are the Hermitian transpose of kets i.e. $\langle\psi| \in \mathcal{H}^\dagger$.

As an illustration, we can represent the following column vectors in Dirac notation

$$|00\ldots0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad |00\ldots1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad |11\ldots0\rangle = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad |11\ldots1\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}.$$

For every ket, there exists a unique bra. Hence, the corresponding dual vectors for these are

$$\begin{aligned} \langle 00\ldots0| &= \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \end{pmatrix} \\ \langle 00\ldots1| &= \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \end{pmatrix} \\ \langle 11\ldots0| &= \begin{pmatrix} 0 & 0 & \ldots & 1 & 0 \end{pmatrix} \\ \langle 11\ldots1| &= \begin{pmatrix} 0 & 0 & 0 & \ldots & 1 \end{pmatrix}. \end{aligned}$$

There is one particular vector to be aware of when using Dirac notation and that is the 0 vector. The 0 vector is never written as $|0\rangle$, which we will see later. Furthermore, Shor's algorithm will represent vectors using digits rather then binary

numbers. As an example, for some computational basis $\{|0\rangle, |1\rangle, \ldots, |N-1\rangle\}$, the N-dimensional vector space is composed of the column vectors

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad |2\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \quad |N-1\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}.$$

Using digits instead of their binary representation makes the notation compact and more manageable when we have very large bases.

We say a basis is orthonormal if every vector is a unit vector, i.e. $\| |v\rangle \| = 1$, and orthogonal to the other vectors. To describe an orthonormal basis we use the Kronecker delta function, $\delta_{i,j}$, where

$$\delta_{i,j} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

Hence, a finite basis, $B = \{|b_i\rangle\} \in \mathcal{H}$, is orthonormal if

$$\langle b_i | b_j \rangle = \delta_{i,j} \qquad \text{for all } b_i, b_j \in B.$$

Every quantum state, $|\psi\rangle \in \mathcal{H}$, with respect to an orthonormal basis $\{|b_i\rangle\}$, can

be represented as

$$|\psi\rangle = \sum_{n=1}^{N} \psi_n |b_n\rangle \qquad \text{for some } \psi_n \in \mathbb{C}.$$

All coefficients of the quantum state, ψ_n, must by assumption obey

$$\psi_1^2 + \psi_2^2 + \ldots + \psi_n^2 = 1.$$

Say we have $|v\rangle$ and $|w\rangle$ where $v, w \in \mathbb{C}$. Then the inner product between $|v\rangle$ and $|w\rangle$ outputs a scalar so that

$$\begin{aligned}\langle v|w\rangle &= \begin{pmatrix} v_1^* & v_2^* & \ldots & v_n^* \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \\ &= v_1^* w_1 + v_2^* w_2 + \ldots + v_n^* w_n \\ &= \sum_{i=1}^{n} v_i^* w_i.\end{aligned}$$

where * denotes the complex conjugate. Another linear operator often performed is the outer product denoted $|v\rangle \langle w|$ and operates as follows.

$$|v\rangle\langle w| = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} \begin{pmatrix} w_1^* & w_2^* & \dots & w_n^* \end{pmatrix}$$
$$= \begin{pmatrix} v_1 w_1^* & v_1 w_2^* & \dots & v_1 w_n^* \\ v_2 w_2^* & v_2 w_2^* & \dots & v_2 w_n^* \\ \vdots & \vdots & \vdots & \vdots \\ v_n w_1^* & v_n w_2^* & \dots & v_n w_n^* \end{pmatrix}.$$

Dirac notation allows us to represent any arbitrary ket as a linear superposition of orthonormal basis states. The superposition principle of quantum mechanics states, if $|v_1\rangle$ and $|v_2\rangle$ are two states in some quantum system, $|\psi\rangle$, then a superposition of the two states is anything of the form $|\psi\rangle = \alpha|v_1\rangle + \beta|v_2\rangle$ where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$ (Nielsen & Chuang, 2000, p. 94). Often α and β are referred to as the complex amplitudes or coefficients of a quantum state. Furthermore, the probability of being in state $|v_1\rangle$ is $|\alpha|^2$ and the probability of being in state $|v_2\rangle$ is $|\beta|^2$.

Two bases which are commonly used in quantum computing are the computational basis, $\{|0\rangle, |1\rangle\}$, and the Hadamard basis, $\{|+\rangle, |-\rangle\}$. The basis vectors of the computational basis can be represented as column vectors

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

This is why we never denote the 0 vector as $|0\rangle$. The column vector representation of the Hadamard basis vectors is

$$|+\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \qquad |-\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right).$$

Throughout the rest of this thesis, we will be interested in combining quantum state vectors and creating larger Hilbert spaces. The tensor product is a way to expand smaller Hilbert spaces into even larger Hilbert spaces. We write the tensor product as follows: if $|V\rangle \in \mathcal{H}_1$ and $|W\rangle \in \mathcal{H}_2$ where the $\dim(|V\rangle) = m$ and the $\dim(|W\rangle) = n$ then the tensor product of $|V\rangle$ and $|W\rangle$ is written as $|V\rangle \otimes |W\rangle \in \mathcal{H}_1 \otimes \mathcal{H}_2$ where $\dim(\mathcal{H}_1 \otimes \mathcal{H}_2) = mn$. The tensor product is often abbreviated as $|VW\rangle$, $|V\rangle\,|W\rangle$ and $|V, W\rangle$.

The tensor product has the following properties. Let $|\psi_1\rangle \in \mathcal{H}_1$ and $|\psi_2\rangle \in \mathcal{H}_2$ then,

(1) For some scalar $c \in \mathbb{C}$,

$$\begin{aligned} c(|\psi_1\rangle \otimes |\psi_2\rangle) &= (c\,|\psi_1\rangle) \otimes (|\psi_2\rangle) \\ &= (|\psi_1\rangle) \otimes (c\,|\psi_2\rangle). \end{aligned}$$

(2) For some $|\phi_1\rangle \in \mathcal{H}_1$,

$$(|\psi_1\rangle + |\psi_2\rangle) \otimes |\phi_1\rangle = |\psi_1\rangle \otimes |\phi_1\rangle + |\psi_2\rangle \otimes |\phi_1\rangle.$$

(3) For some $|\phi_2\rangle \in \mathcal{H}_2$,

$$|\psi_1\rangle \otimes (|\psi_2\rangle + |\phi_2\rangle) = |\psi_1\rangle \otimes |\psi_2\rangle + |\psi_1\rangle \otimes |\psi_2\rangle .$$

With the above tensor product properties we can now define entanglement. A quantum state, $|\psi\rangle$, is entangled if there are no single states $|v\rangle$ and $|w\rangle$ such that $|\psi\rangle = |v\rangle \otimes |w\rangle$ (Nielsen & Chuang, 2000, p. 96). As an illustration of entanglement, we will show the basis vector, $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is entangled.

Proof:

Suppose we have two quantum states $|\phi_1\rangle = \alpha |0\rangle + \beta |1\rangle$ and $|\phi_2\rangle = \gamma |0\rangle + \delta |1\rangle$ where $\alpha, \beta, \gamma, \delta \in \mathbb{C}$. We need to show that $|\psi\rangle \neq |\phi_1\rangle |\phi_2\rangle$ so we will proceed with a proof by contradiction. Assume $|\psi\rangle = |\phi_1\rangle |\phi_2\rangle$. Then

$$\begin{aligned} |\psi\rangle &= (\alpha |0\rangle + \beta |1\rangle)(\gamma |0\rangle + \delta |1\rangle) \\ &= \alpha\gamma |00\rangle + \alpha\delta |01\rangle + \beta\gamma |10\rangle + \beta\delta |11\rangle . \end{aligned}$$

For this to be true, $\gamma\delta = 0$. When $\gamma = 0$, $\alpha\gamma = 0 \neq \frac{1}{\sqrt{2}}$. Similarly, if $\delta = 0$ then $\beta\delta = 0 \neq \frac{1}{\sqrt{2}}$. This means $|\psi\rangle$ can not be decomposed into a product of the two states $|\phi_1\rangle$ and $|\phi_2\rangle$ which shows $|\psi\rangle$ is entangled.

MEASUREMENT

The quantum systems we have been describing are closed, meaning they do not suffer from unwanted interactions from the surrounding environment. This is great, however far from satisfying. More often then not, experimentalists want to know what is happening inside the quantum system. Just like in our example of entanglement, both observers wanted to know which color ball they were holding. In order to see inside a closed quantum system, we must make a measurement. Note that measuring the state of a quantum system is sometimes referred to as observing the state of a quantum system.

A major consequence of measuring the state of a quantum system is that the result is irreversible. Once a measurement is made, the state of the quantum system can never return to its original state prior to the measurement. As we will see in Shor's Algorithm, if you need to make a measurement on the state of a quantum system, you better hope the result is what you want because there is no way of going back. When we measure the state of an unknown closed quantum system, we change that state in some indeterminate way. This is called quantum indeterminacy (Jha et al.) Quantum indeterminacy is the basis of the Heisenberg Uncertainty Principle and the No Cloning Theorem. It is often exploited to detect any eavesdroppers as well as determine how much information has been compromised or intercepted, for example in Quantum Key Distribution (Rubenok, 2013)

INTRODUCTION TO SHOR'S ALGORITHM

Now that we have the necessary quantum mechanical foundations and phenomena as well as the notation to describe quantum systems, we will elaborate on certain subject matter which are crucial to Shor's algorithm and hopefully elucidate the algorithm. The prime factorization problem Shor's algorithm solves is

Given a integer N, find exactly two primes p and q such that $N = pq$.

The prime factorization problem used here is intended for factoring large integers. The quantum part of Shor's algorithm has a run time of

$$\mathcal{O}\left((\log N)^2 \cdot (\log(\log N))\right)$$

which is significantly more efficient than the GNFS algorithm which has a run time of

$$\mathcal{O}\left(exp\left((\log N)^{1/3} \cdot (\log(\log N))^{2/3}\right)\right).$$

In other words, Shor's algorithm solves the prime factorization problem for large integers in polynomial time by finding the period, r, of some superposed periodic function $f(x) = a^x \pmod N$ and then applying classical algorithms to find a factor corresponding to the period. If r is even, then for some integer x, we can solve $\gcd(x^{r/2} - 1, N)$ and $\gcd(x^{r/2} + 1, N)$ which will ideally give a factor of N. If r is not even then $x^{r/2} \pm 1$ will not be an integer which means we do not have a valid prime factor of N. When this occurs, we can repeat the process and solve using a different

r until we get the right factors. The most challenging part of this prime factorization problem, which Shor was able to execute beautifully, is finding the period.

Shor's Algorithm can be divided into two parts. The first part takes the prime factorization problem and turns it into the period finding problem which can be implemented on a classical computer. The second part of the algorithm finds the period using the Quantum Fourier Transform. It would seem that both parts could be implemented on a classical computer. However, the intent is to factor huge positive composite integers. Applying a Discrete Fourier Transform to such a large set would be impractical and inefficient. Shor's Algorithm relies on a quantum computer's ability to compute simultaneous operations at once on quantum states, i.e. states in superposition. This massive parallel computing is derived from entanglement and superposition. So instead of performing the DFT multiple times on a classical computer in hopes of finding the period, we can compute the QFT once for all states which will return the period with high probability.

QUANTUM FOURIER TRANSFORM

In mathematics, sometimes we are not able to find a solution to a problem simply by how the problem is set up. When this happens, we can apply some sort of transformation to the problem. By doing so we can transform the problem into another problem where a solution is known. The Discrete Fourier Transform (DFT) is a great example of this. The DFT takes some vector of input data, applies a transformation on the vector, then outputs the transformed data. Commonly in signal processing this would be like taking a signal which resides in some time domain,

apply a transformation to that signal and the result outputs all of the frequencies within that signal i.e. the output of the signal is now in the frequency domain. We aren't actually changing the data itself. We are simply transforming it.

Suppose we have a vector of length N with input values $x_0, \ldots, x_{N-1} \in \mathbb{C}$. The DFT takes these input values and outputs a vector of transformed values call them $y_0, \ldots, y_{N-1} \in \mathbb{C}$ by performing the following transformation (Riley et al., 2006 p. 462)

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i j k/N}.$$

Let the roots of unity be written as $\omega = e^{2\pi i/N}$ (Chen, 2001, p. 24). Then the DFT has the following matrix representation

$$\begin{pmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \\ \vdots \\ y(N-1) \end{pmatrix} = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & 1 & \ldots & 1 \\ 1 & w & w^2 & w^3 & \ldots & w^{N-1} \\ 1 & w^2 & w^4 & w^6 & \ldots & w^{N-2} \\ 1 & w^3 & w^6 & w^9 & \ldots & w^{N-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & w^{N-1} & w^{N-2} & w^{N-3} & \ldots & w \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ \vdots \\ x(N-1) \end{pmatrix}$$

We illustrate the DFT by using the Fast Fourier Transform to compute the DFT on a continuous sinusoidal waveform (see Figure 3 and Figure 4). The FFT decomposes a DFT into several DFT's on smaller intervals (Smith, 1997, p. 157). This is an incredibly efficient algorithm for computing the DFT on a classical computer.

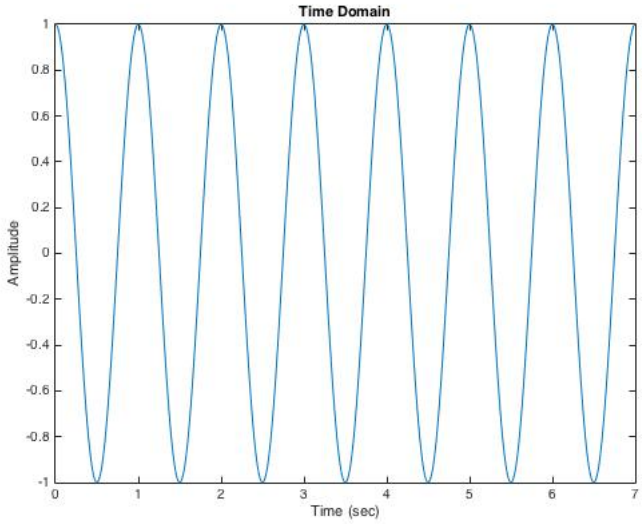


Figure 3: Sinusoidal Waveform in Time Domain

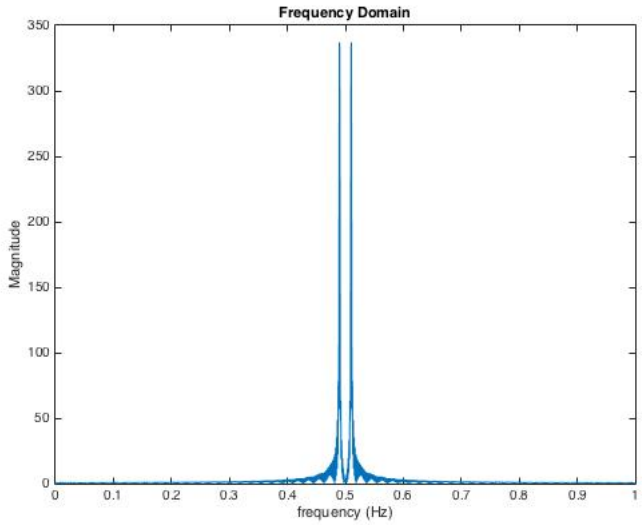


Figure 4: Approximate Spectrum of Sinusoidal Waveform

The Quantum Fourier Transform (QFT) is the same linear transformation as the DFT applied to quantum bits. More specifically, the QFT applies a DFT to the amplitudes of some quantum state. As we will later see, Shor's Algorithm exploits this transformation in order to solve the factorization problem. The QFT acting on some orthonormal basis $|0\rangle, \ldots, |N-1\rangle$ is a linear operator written as

$$QFT(|x\rangle) = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} e^{2\pi i x k/N} |k\rangle .$$

We will prove that the QFT is a unitary transformation (Nielson & Chuang, 2000). Let F denote the QFT. We will show F is a unitary operator, i.e. $F^\dagger F = FF^\dagger = I$. Write $|\tilde{\psi}\rangle = F|\psi\rangle$ for some quantum state $|\psi\rangle$. Here $|\tilde{\psi}\rangle$ represents the quantum state after the transformation has been applied. We can write the unitary operator, F, as

$$\begin{aligned} F &= \frac{1}{\sqrt{N}} \sum_{j'=0}^{N-1} \sum_{k'=0}^{N-1} e^{2\pi i j' k'/N} |k'\rangle \langle j'| \\ &= \frac{1}{\sqrt{N}} \sum_{j',k'=0}^{N-1} e^{2\pi i j' k'/N} |k'\rangle \langle j'| . \end{aligned}$$

The Hermitian transpose of our unitary operator, $F^\dagger$, can be written as

$$\begin{aligned} F^\dagger &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} e^{-2\pi i j k/N} |j\rangle \langle k| \\ &= \frac{1}{\sqrt{N}} \sum_{j,k=0}^{N-1} e^{-2\pi i j k/N} |j\rangle \langle k| . \end{aligned}$$

And now for the proof that F is indeed a unitary transformation.

$$\begin{aligned}
F^\dagger F &= \frac{1}{\sqrt{N}} \sum_{j,k=0}^{N-1} e^{-2\pi ijk/N} \left|j\right\rangle \left\langle k\right| \frac{1}{\sqrt{N}} \sum_{j',k'=0}^{N-1} e^{2\pi ij'k'/N} \left|k'\right\rangle \left\langle j'\right| \\
&= \frac{1}{N} \sum_{j,k=0}^{N-1} \sum_{j',k'=0}^{N-1} e^{-2\pi ijk/N} \left|j\right\rangle \left\langle k\right| e^{2\pi ij'k'/N} \left|k'\right\rangle \left\langle j'\right| \\
&= \frac{1}{N} \sum_{j,k=0}^{N-1} \sum_{j',k'=0}^{N-1} e^{2\pi i(j'k'-jk)/N} \left|j\right\rangle \left\langle k|k'\right\rangle \left\langle j'\right| \\
&= \frac{1}{N} \sum_{j,k,j'=0}^{N-1} e^{2\pi i(j'-j)k/N} \left|j\right\rangle \left\langle j'\right|
\end{aligned}$$

Note if $j = j'$ then $e^{2\pi i(j'-j)/N} = e^0 = 1$. If $j \neq j'$ then $e^{2\pi i(j'-j)/N} = \left(e^{2\pi i}\right)^{(j'-j)/N}$. Recall that $e^{2\pi i} = 1$. Hence $\left(e^{2\pi i}\right)^{(j'-j)/N} = 1^{(j'-j)/N} = 1$. This means we can write $F^\dagger F$ as

$$F^\dagger F = \sum_{j,k,j'=0}^{N-1} \left|j\right\rangle \left\langle j'\right| .$$

Any arbitrary quantum state, $\left|\psi\right\rangle$, can be represented as a linear combination of basis vectors $\{\left|N\right\rangle : \left|0\right\rangle, \ldots, \left|N-1\right\rangle\}$

$$\left|\psi\right\rangle = \sum_{N=0}^{N-1} \phi_N \left|N\right\rangle$$

where $\sum_{N=0}^{N-1} \phi_N^2 = 1$. Since the basis is orthonormal, then the expansion coefficients

can be written as

$$\phi_N = \langle N|\psi\rangle .$$

This means the quantum state can be written as

$$\begin{aligned}
|\psi\rangle &= \sum_{N=0}^{N-1} \phi_N |N\rangle \\
&= \sum_{N=0}^{N-1} |N\rangle \phi_N \\
&= \sum_{N=0}^{N-1} |N\rangle \langle N|\psi\rangle \\
&= \sum_{N=0}^{N-1} |\psi\rangle .
\end{aligned}$$

The only way this is true is if $\sum_{N=0}^{N-1} |N\rangle \langle N| = I$. We call this representation of the identity the completeness relation or the resolution of the identity. By the completeness relation

$$F^\dagger F = \sum_{j,j'=0}^{N-1} |j\rangle \langle j'| = I.$$

Showing that $FF^\dagger = I$ is similar so we omit that here. Therefore we can conclude that F is a unitary operation. Furthermore, because F represented the QFT, we can conclude that the QFT is a unitary transformation. This is important because a quantum circuit implementing the QFT can run in reverse to implement the Inverse Quantum Fourier Transform (IQFT) if needed.

CONTINUED FRACTIONS

The continued fraction algorithm is an incredibly important step in Shor's algorithm. By approximating real numbers as rationals, we can obtain a sequence of convergents to a continued fraction. Within this sequence of convergents, we will discover the period of the QFT that we seek in Shor's algorithm. Once we find the period using the continued fractions algorithm, we use the exponent factorization method to solve the factorization problem.

For any $x \in \mathbb{Q}$ and $x \geq 0$, we can write x as a finite continued fraction. A finite continued fraction is a collection of partial quotients, or simply quotients, $[a_0, a_1, \ldots, a_n] \in \mathbb{Z}^+$ written as

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}}}$$

Sometimes we denote our rational number, x, as $[a_0, a_1, \ldots, a_n]$ (i.e. $x \equiv [a_0, a_1, \ldots, a_n]$) (Nielsen & Chuang, 2000, p.635 - 637). Also, if there is no loss of ambiguity, finite continued fractions are simply called continued fractions. We can calculate convergents of the continued fraction as follows. Let p_n and q_n be real numbers so that p_0,

p_1, q_0 and q_1 are

$$p_0 = 0$$

$$p_1 = 1 + a_0 a_1$$

$$q_0 = 1$$

$$q_1 = a_1.$$

We can inductively write the remaining p_n and q_n as

$$p_n = a_n p_{n-1} + p_{n-2}$$

$$q_n = a_n q_{n-1} + q_{n-2}.$$

The convergents of a continued fraction are of the form

$$\frac{p_n}{q_n} = \frac{a_n p_{n-1} + p_{n-2}}{a_n q_{n-1} + q_{n-2}}.$$

The continued fraction algorithm will terminate after a finite number of iterations (Hardy & Wright, 2008, p. 166). As an example, we will find the continued fraction representation of $\frac{181}{101}$ as well as the sequence of convergents for this rational number. First we will use Euclid's algorithm to find the partial quotients. In other words,

find the greatest common divisor of 181 and 101 as follows

$$\begin{aligned} 181 &= 1 \cdot 101 + 80 \\ 101 &= 1 \cdot 80 + 21 \\ 80 &= 3 \cdot 21 + 17 \\ 21 &= 1 \cdot 17 + 4 \\ 17 &= 4 \cdot 4 + 1 \\ 4 &= 4 \cdot 1 + 0. \end{aligned}$$

The partial quotients of the continued fraction for $\frac{181}{101}$ are read off from the algorithm as $[1, 1, 3, 1, 4, 4]$. Therefore the continued fraction representation of $\frac{181}{101}$ is

$$\frac{181}{101} = 1 + \cfrac{1}{1 + \cfrac{1}{3 + \cfrac{1}{1 + \cfrac{1}{4 + \cfrac{1}{4}}}}}.$$

Using the partial quotients we find the convergents, $\frac{p_n}{q_n}$, of this continued fraction to be

$$\left\{0,\ 1,\ \frac{3}{4},\ \frac{4}{5},\ \frac{19}{24},\ \frac{80}{101}\right\}.$$

If $x \in \mathbb{R}\backslash\mathbb{Q}$, a continued fraction expansion exists but it is infinite. For example, the continued fraction representation of π is infinite. Although important, We will only be handling finite continued fractions in Shor's algorithm.

EXPONENT FACTORIZATION

Once we derive the period from the sequence of convergents, which will be discussed in greater detail in the steps of Shor's algorithm, we apply the exponent factorization method to find the greatest common divisor of the integer n to be factored with another integer. The exponent factorization method we used is directly from (Trappe & Washington, 2006) and works as follows.

Suppose we have some exponent, which in Shor's algorithm this will be the period r, where $r > 0$ such that for some $a \in \mathbb{N}$ we have $a^r \equiv 1 \pmod{n}$. Write the exponent, r, as a product of an odd integer, m, and some power of two, 2^k, so that $r = 2^k m$. Let $b_0 \equiv a^m \pmod{n}$ and $b_{u+1} \equiv b_u^2 \pmod{n}$ for $0 \leq u \leq k-1$.

1. If $b_0 \equiv 1 \mod n$, stop. The factorization method has failed to find a factor of n.

2. If, for some u, $b_u \equiv -1 \pmod{n}$, stop. The method has failed.

3. If $b_{u+1} \equiv 1 \pmod{n}$ but $b_u \not\equiv \pm 1 \pmod{n}$, then $\gcd(b_u - 1, n)$ gives a nontrivial factor of n.

QUANTUM BITS & HADAMARD TRANSFORMATION

Quantum Computers are fundamentally different than classical/digital computers. They utilize the quantum mechanical phenomena superposition and entangle-

ment to perform operations. Classical computers can only accept data if it is encoded into binary digits or bits and then perform operations using logic gates such as AND and OR. Bits can only ever be 0 or 1 at any given time. On the other hand, quantum computers can accept encoded data which is in a superposition of states 0 and 1. We refer to these superposition bits as quantum bits or qubits for short. The most common qubit used in quantum computing is the quantum state, $|\psi\rangle$, written with respect to the computational basis $\{|0\rangle, |1\rangle\}$ as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ where where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. It is important to note that qubits are only in a state of superposition before any measurement has been made. Once a measurement is made on the quantum state, the superposed state of a qubit collapses to $|0\rangle$ or $|1\rangle$. We never actually see that a qubit is in a superposition because by simply measuring the state, we cause an effect which collapses the state.

Gate operations applied on quantum bits are unitary transformations. One important example is the Hadamard transformation. The Hadamard transform performs an orthogonal, symmetric and linear operation on 2^n complex or real numbers. The Hadamard transform is a multidimensional $2^n \times 2^n$ DFT normalized to be unitary. We can find the values for any $2^n \times 2^n$ Hadamard matrix by using the following equation

$$(H_n)_{i,j} = \frac{1}{\sqrt{2^n}}(-1)^{i \cdot j}$$

where i and j are the i-th row and j-th column entry in a $2^n \times 2^n$ matrix. For example, the Hadamard transform used most in quantum computing, and which will be used in Shor's algorithm, is the matrix

$$H_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

which is exactly a DFT of size 2. Shor's algorithm uses the Hadamard transform as an initial step in the algorithm because the Hadamard transform maps n-qubits which have been initialized with $|0\rangle$ e.g. $|1,0\rangle + |2,0\rangle + \ldots + |n,0\rangle$, to a superposition of $2 \cdot n$ orthogonal states of equal weight with respect to the computational basis $\{|0\rangle, |1\rangle\}$. In other words, the Hadamard transform maps the computational basis to the Hadamard basis as follows

$$|0\rangle \rightarrow \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$
$$|1\rangle \rightarrow \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

Thus when we apply the Hadamard transform to a quantum state in superposition, $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, we obtain the quantum state, $|\psi'\rangle$, written as

$$\begin{aligned} |\psi'\rangle &= \alpha\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) + \beta\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\ &= \left(\frac{\alpha+\beta}{\sqrt{2}}\right)|0\rangle + \left(\frac{\alpha-\beta}{\sqrt{2}}\right)|1\rangle. \end{aligned}$$

The Hadamard transform essentially transforms the range of states that a quantum computer can be in. By doing so we can take short cuts which can not be done on a classical computer. This will allow us to do some computations faster.

In Shor's algorithm, we will want to apply the Hadamard gate to qubits in some initial state. We know that the Hadamard gate maps $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$. A schematic representation of applying the Hadamard gate, H, to three qubits, $|0\rangle$, is illustrated in figure 5.

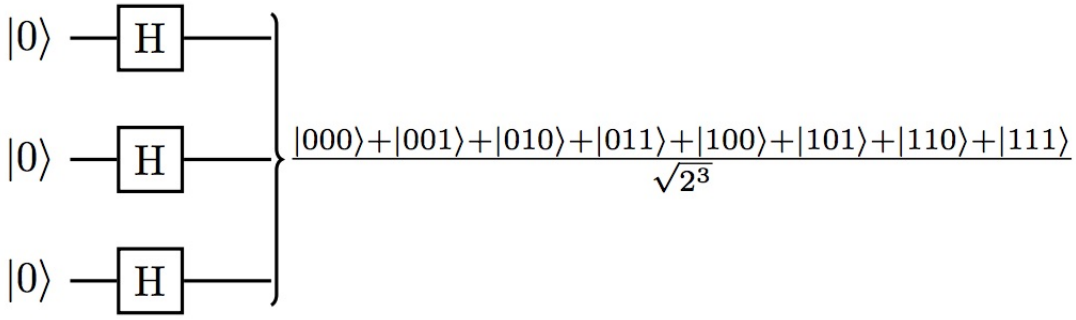


Figure 5: Hadamard Transform on 3 qubits

The schematic tells us once we apply a Hadamard gate to three qubits we obtain a superposition of eight qubits. In general, the Hadamard gate transforms some ket $|x\rangle$ into a superposition of $(-1)^x |x\rangle + |1 - x\rangle$.

There is another way to visualize an m-qubit Hadamard transformation on some state, which will also be helpful in Shor's algorithm (Kaye, Laflamme & Mosca, 2007, p. 100). Let $|0\rangle^{\otimes m}$ denote the tensor product of m-qubits each in the state $|0\rangle$ i.e.

$$|0\rangle^{\otimes m} = \underbrace{|0\rangle \otimes |0\rangle \otimes \ldots \otimes |0\rangle}_{m \text{ times}}.$$

An m-qubit Hadamard transformation is denoted as $H^{\otimes m}$. Therefore m-qubit

Hadamard transformations is written as

$$H^{\otimes m}\left|0\right\rangle^{\otimes m} = \frac{1}{\sqrt{2^m}} \underbrace{\left(\left|0\right\rangle + \left|1\right\rangle\right) \otimes \left(\left|0\right\rangle + \left|1\right\rangle\right) \otimes \ldots \otimes \left(\left|0\right\rangle + \left|1\right\rangle\right)}_{m \text{ times}}.$$

If we expand out this tensor product we will find that the transformation can be written as the sum

$$H^{\otimes m}\left|0\right\rangle^{\otimes m} = \frac{1}{\sqrt{2^m}} \sum_{x \in \{0,1\}^m} \left|x\right\rangle.$$

SHOR'S ALGORITHM

If we want a quantum computer to physically act like a classical computer, the quantum computer needs a way to keep track of any computations that occur with quantum bits. To do so we use quantum memory registers. A memory register in a classical computer contains data, i.e. bits, and records any operations performed on those bits. A quantum memory register does the same thing for quantum bits (Mollin, 2001, p. 28). We can think of quantum memory registers as just a string of qubits (Vries, 2012, p. 20).

Before implementing Shor's algorithm (Shor, 1994), create a quantum memory register composed of two registers, an input register call it register 1 and an output register call it register 2. The state of the qubits in register 1 will be initialized to some superposed quantum state and the state of the qubits in register 2 will contain the values of the periodic function $f(x) = \left|a^x \pmod n\right\rangle$. According to (Kendon &

Munro, 2005), if register 2 has $2n$ qubits, we have a high enough accuracy to find the period from one measurement.

Shor's Algorithm

1. Pick a composite integer to factor and call it n.

2. Find an integer m such that $n^2 \leq 2^m < 2n^2$.

3. Choose a random integer a such that $1 < a < n$ and $\gcd(a, n) = 1$.

4. Create two registers. Initialize register 1 and register 2 of size 2^m both to state $|0\rangle$ so that the composite system has the state $|\psi\rangle = |0\rangle\,|0\rangle$. For now, we will use this tensor product notation to distinguish between registers, e.g. if we initialize register 1 to $|a\rangle$ and register 2 to $|b\rangle$, then the composite system has the state $|\psi\rangle = |a\rangle\,|b\rangle$. We can think of a composite quantum system as the tensor product of multiple qubits e.g. $|\psi_1\rangle \otimes |\psi_2\rangle \otimes |\psi_3\rangle \otimes \ldots \otimes |\psi_m\rangle$.

5. Apply the Hadamard gate, H, to each of the m qubits in register 1 to create a superposition of states. As we've shown earlier, when we apply a Hadamard gate to m qubits we obtain a superposition of 2^m qubits normalized by $\frac{1}{\sqrt{2^m}}$. So applying the Hadamard gate to m qubits in register 1 leaves the computer in the superposed quantum state

$$|\psi\rangle = \frac{1}{\sqrt{2}} \sum_{x=0}^{2^m-1} |x\rangle\,|0\rangle = \frac{1}{\sqrt{2}}[|0\rangle + |1\rangle + |2\rangle + \ldots + |2^m - 1\rangle]\,|0\rangle\,.$$

6. Compute $f(x) = a^x \pmod{n}$ for $0 \leq x < 2^m$ which will be

$$\begin{aligned} |\psi'\rangle &= \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} |x\rangle \, |a^x \bmod n\rangle \\ &= \frac{1}{\sqrt{2^m}} \Big[\, |0\rangle \, |f(0)\rangle + |1\rangle \, |f(1)\rangle + \ldots + |2^m - 1\rangle \, |f(2^m - 1)\rangle \, \Big]. \end{aligned}$$

Leave the results in register 2.

7. Make a measurement on register 2. By doing so we will discover that register 2 will be in some base state $|a^x \pmod{n}\rangle$. By measuring register 2 we are collapsing the superposed quantum state into a smaller superposed quantum state.

8. Apply the QFT to our newly measured quantum state $|x, f(u)\rangle$ as follows

$$QFT(|x\rangle) = \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} e^{2\pi i x k / 2^m} \, |k\rangle \, .$$

9. From this we obtain the sum

$$\frac{1}{\sqrt{W}} \sum_{x=0}^{2^m-1} g(k) \, |k\rangle$$

where W is the total number of qubits in our measured quantum state which was made in step 7 of this procedure. The function $g(k)$ is given by

$$g(k) = \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} e^{2\pi i x k / 2^m}.$$

which is just the DFT of some binary sequence.

10. Choose one of the 2^m integers which has a high probability of being a peak in our QFT plot. Note that if this were actually implemented on a quantum computer we would not have all the 2^m. After we make a measurement and apply the QFT, the quantum computer will only output a single point from the 2^m integers with high probability of this point being near a peak. As we will see in the plot of the QFT in our example, it is easy to pick the right point or peak since we implemented this classically. It is important to select, or have the quantum computer output, one of the 2^m integers which is closest to being a peak because we will have a higher probability of obtaining the period of our periodic function. This in return will allow us to find a factor of n with high probability. We will show results for using integers close to a peak and integers with low probabilities of being a peak.

11. Using the integer selected in step 10, we need to create a continued fraction. If P is the value we selected then find the continued fraction for $\frac{P}{2^m}$. Once we obtain the continued fraction, find the convergent values, $[b_0, b_1, \ldots, b_n]$ by calculating

$$\frac{p_n}{q_n} = \frac{b_n p_{n-1} + p_{n-2}}{b_n q_{n-1} + q_{n-2}}$$

where $p_0 = b_0$, $p_1 = b_1 b_0 + 1$, $q_0 = 1$ and $q_1 = b_1$. This will output a sequence of fractions. Within that sequence, find the last denominator which is less then n. This denominator is the period, r, which is what we've been solving for.

Check that $a^r \equiv 1 \mod n$.

12. Apply the exponent factorization method to factor n using the new found period r. Write the period as $r = 2^k m$ where $k \geq 0$ and m is some odd integer. If the exponent factorization method fails, then choose a different random variable a in step 3 of this procedure, and repeat steps 4 through 12 until a factor of n is found.

SIMULATION OF SHOR'S ALGORITHM

Now that we know the procedure, we will demonstrate Shor's algorithm by factoring 33 into its product of primes 3 and 11. Set the initial parameters to $n = 33$, the random variable $a = 7$ and the number of qubits $m = 11$. Recall that the random variable must satisfy $1 < a < n$ and the number of qubits is derived from $n^2 \leq 2^m < 2n^2$. The next step is where we start to deviate away from the quantum phenomenon which creates a massive parallelism for computing. Quantum or not we need to compute the function $f(x) = |a^x \pmod n\rangle$ for $1 \leq x \leq 2048$. Note that MATLAB starts the indexes at 1 for summations whereas Shor's algorithm initiates the index from 0 and runs to $2^m - 1$. Here we run from 1 to 2048.

Because we are restricted to the physical properties of classical mechanics, we do not initialize the input register to a superposition of quantum states. We physically can not do that if we don't have access to the properties of quantum mechanics. The benefit to finding all $f(x)$'s using a quantum computer is that this computation would just happen once. Meaning, we are able to perform all 2^m modular exponentiation

functions simultaneously. Once we compute $f(x) = |a^x \pmod n\rangle$, we obtain the quantum state

$$|\psi\rangle = \frac{1}{\sqrt{2048}}\Big[\,|0,1\rangle + |1,7\rangle + |2,16\rangle + |3,13\rangle + |4,25\rangle + \\ |5,10\rangle + |6,4\rangle + |7,28\rangle + |8,31\rangle + \\ |9,19\rangle + |10,1\rangle + \ldots + |2047,28\rangle + |2048,31\rangle\,\Big].$$

The way we bypassed handling a superposed quantum state in a digital register is by representing $|\psi\rangle$ as one long column vector of length $(2^{11})^2 = 4194304$. Next we need to measure the quantum state (or column vector) $|\psi\rangle$. When we make a measurement on a quantum computer, we have no way of knowing which state the closed quantum state will collapse to. However, we do know that there is equal probability of collapsing the quantum state to one of kets within the quantum state. This is because all kets are weighted equally by the fraction $\frac{1}{\sqrt{2^m}}$.

In order to simulate the effect of a measurement collapsing a quantum state, we explicitly defined $f(x)$ to one of the values in its periodic sequence. So for this simulation, we defined our new post-measurement quantum state, $|\psi'\rangle$, to be all kets which are 7 modulo 33. That means we can write $|\psi'\rangle$ as

$$|\psi'\rangle = \frac{1}{\sqrt{204}}[|1,7\rangle + |11,7\rangle + |21,7\rangle + |31,7\rangle + \\ |41,7\rangle + |51,7\rangle + \ldots + |2031,7\rangle + |2041,7\rangle].$$

Recall that the tensor product between qubits can be represented as $|a, b\rangle$ or $|a\rangle\,|b\rangle$ for some qubit a in register 1 and some qubit b in register 2. This notation helps us distinguish between the values contained in each register. Since all of the important outputs of $f(x)$ are stored in register 2, we only care to make a measurement on register 2. For notational purposes we drop the second half of each ket in $|\psi'\rangle$ so that

$$|\psi'\rangle = \frac{1}{\sqrt{204}}[|1\rangle + |11\rangle + |21\rangle + |31\rangle + |41\rangle + |51\rangle + \ldots + |2031\rangle + |2041\rangle].$$

If we were using a quantum computer, the next step would be to apply the Quantum Fourier Transform to $|\psi'\rangle$ and then compute the $g(k)$ values. We, on the other hand, applied the Fast Fourier Transform to the modular exponentiation functions and plotted the absolute values of $g(k)$. The peaks of the absolute values of $g(k)$ should correspond to the frequency of the DFT sequence. This should be around $2^m/r = 2^{11}/10 = 2048/10$. We call $2^m/r$ the fundamental frequency. We know that the period is 10 from computing the convergents of the continued fraction which are

$$\{0,\ 1,\ \frac{9}{10},\ \frac{226}{251},\ \frac{235}{261},\ \frac{461}{512}\}.$$

As we see in Figure 6, the sharp peaks do occur every $k = 204.8$. Figure 7 is the same information obtained in Figure 6, just scaling the y-axis to better show the local minima. More plots of the absolute value of $g(k)$ for all random variables, a,

such that $1 \leq a < 33$ are shown in Appendix A.

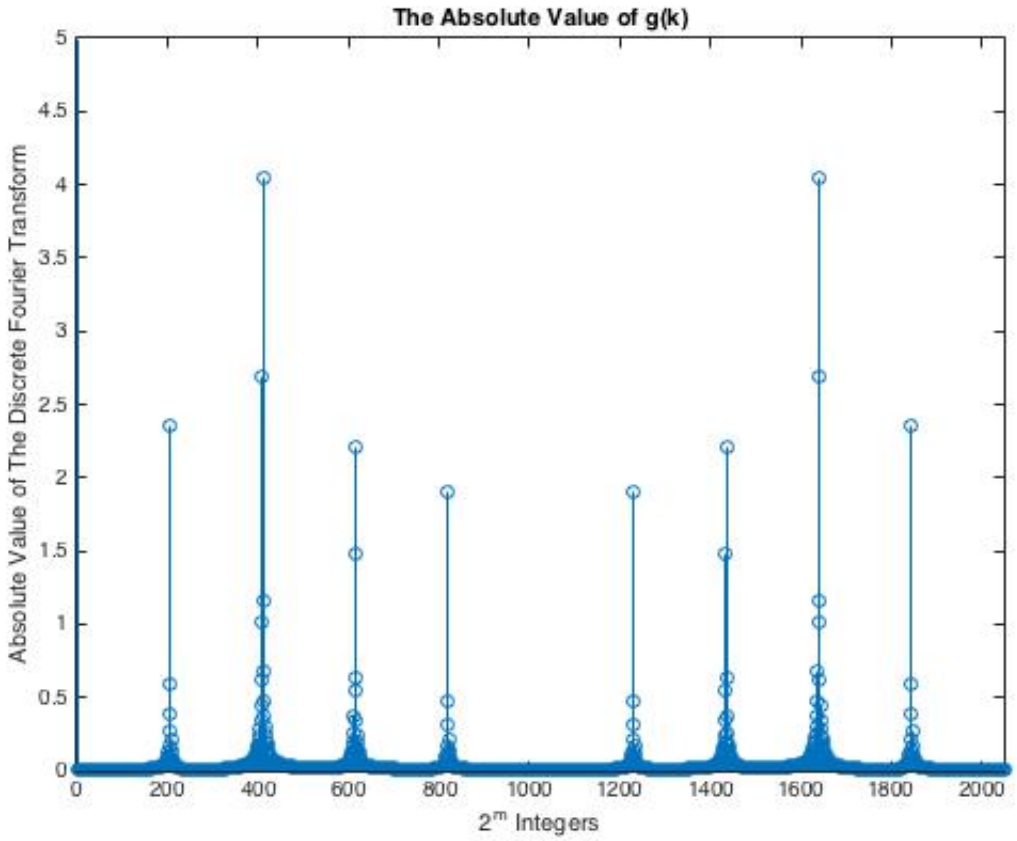


Figure 6:

Result of factoring 33 into a product of 3 and 11 using random variable a = 7 with the period of the function being 10.

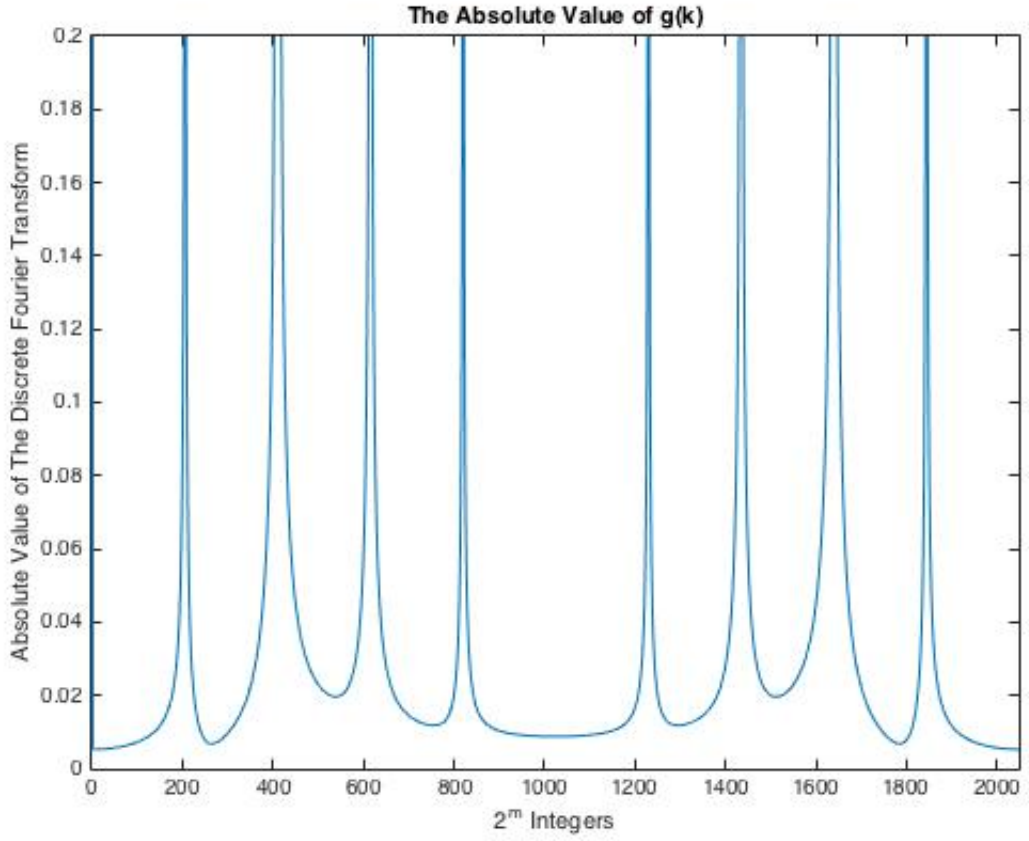


Figure 7:

Local minima from the result of factoring 33 into a product of 3 and 11 using random variable a = 7 with the period of the function being 10.

Each plot of the absolute value of $g(k)$ will be symmetric about 2^{m-1} as a result of the symmetric property of the DFT. These plots of the absolute value of $g(k)$ are a great illustration because we can find where all the peaks are and select those peaks directly to find the period with high probability. On the other hand, a quantum computer would only output one integer. We would have little information regarding other extrema. Peter Shor showed there is high probability of obtaining $\frac{k}{2^m}$ with

$$\left|\frac{k}{2^m} - \frac{\text{peak}}{r}\right| < \frac{1}{2^{m+1}} < \frac{1}{2n^2}$$

where "peak" is a peak value from the absolute value of $g(k)$ (Shor, 1997).

RESULTS OF SIMULATION

For our initial run of Shor's algorithm, with $m = 11$ and the random variable set at $a = 7$, the peaks and corresponding values are shown in the table below.

Peak Location (2^m)	Absolute Value of $g(k)$
206	2.3476
411	4.0452
615	2.2101
820	1.9067
1230	1.9067
1435	2.2101
1639	4.0452
1844	2.3276

We implemented our modified Shor's algorithm on all possible random variables for 33, that is for all $1 \leq a < 33$. The quantum state measured at 7 remains unchanged for each run. We chose select integers or peak values to compute the period. The gcd column in figure 8 shows the factor our algorithm computed, if it did not fail at any step in the procedure. The issues and successes we obtained for

each run with different a values are shown in figure 8.

After running through each random variable, our results show that 25% of the time we actually obtain a factor of n being 3 or 11. If we include 1, which seems trivial, as a factor of n, then 37.5% of the runs will have a successful output. Our algorithm selected a peak value by finding all local maxima which occurred in $|g(k)|$ then selecting the last value of the sequence. There was no particular reason why this peak was chosen other than convenience.

Most of the failures we had with certain random variables occurred during the exponent factorization step of Shor's algorithm. When we computed $b_{u+1} \pmod{n}$, we found this to be equivalent to 12 or 22. Therefore, we were not able to obtain a factor of n since 12 and 22 are definitely not equivalent to 1 $\pmod{n}$. Another issue in the exponent factorization method was when $b_1 \pmod{n}$ was not equivalent to 1. Thus, we could not obtain a factor of n. A plot of the absolute value of $g(k)$ when the random variable chosen could not successfully factor n is provided in figure 9.

Random Variable (a)	Factor of 33 Found	Period	Chosen Peak	Probability at Peak	Issues or Success
1	none	N/A	none	none	index unobtainable
2	1	10	1844	0.5545	success
3	none	5	1639	none	mod(b(u+1),n) ≠ 1 (==12)
4	none	N/A	1639	none	mod(b(1),n) ≠1
5	11	10	1844	0.271	success
6	none	10	1844	0.2498	mod(b(u+1),n) ≠ 1 (==12)
7	3	10	1844	0.0663	success
8	1	10	1844	0.0229	success
9	none	5	1639	0.0671	mod(b(u+1),n) ≠ 1 (==12)
10	none	N/A	1025	none	mod(b(1),n) ≠1
11	none	4	1025	none	mod(b(u+1),n) ≠ 1 (==22)
12	none	N/A	none	none	index unobtainable
13	3	10	1844	0.0898	success
14	11	10	1844	0.0395	success
15	none	5	1639	0.238	mod(b(u+1),n) ≠ 1 (==12)
16	none	N/A	1639	none	mod(b(1),n) ≠1
17	1	N/A	1844	0.5535	success
18	none	10	1844	0.707	mod(b(u+1),n) ≠ 1 (==12)
19	3	10	1844	0.0662	success
20	11	10	1844	0.2702	success
21	none	4	1025	none	mod(b(u+1),n) ≠ 1 (==12)
22	none	N/A	none	none	Index unobtainable
23	none	4	none	1.8264	mod(b(1),n) ≠1
24	none	10	1844	0.2492	mod(b(u+1),n) ≠ 1 (==12)
25	none	N/A	1639	none	mod(b(1),n) ≠1
26	11	10	1844	0.0394	success
27	none	N/A	1639	none	mod(b(u+1),n) ≠ 1 (==12)
28	3	10	1844	0.0895	success
29	1	10	1844	0.0226	success
30	none	N/A	1844	0.0708	mod(b(u+1),n) ≠ 1 (==12)
31	none	N/A	1639	0.0163	mod(b(1),n) ≠1
32	none	N/A	1025	none	mod(b(1),n) ≠1
33	none	N/A	none	none	Index unobtainable

Figure 8:

Results for each random variable $1 \leq a < 33$.

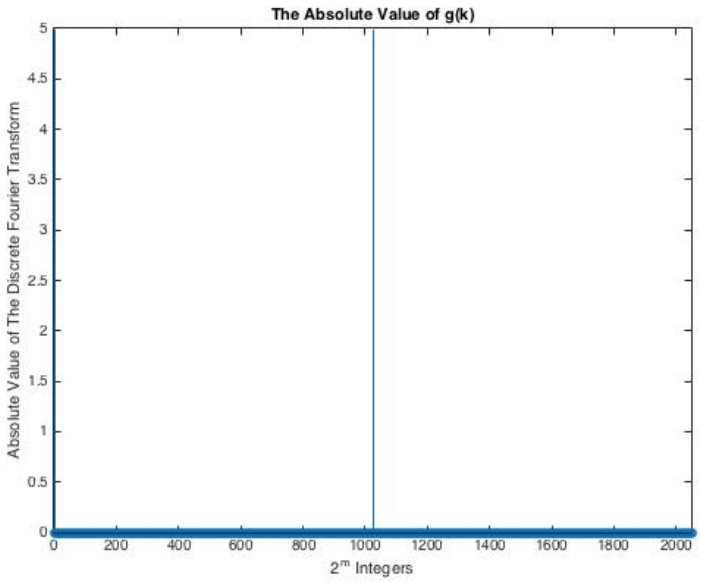


Figure 9:

Failed run of Shor's algorithm with random variable 23.

INVERSE QUANTUM FOURIER TRANSFORM

As mentioned earlier in this thesis, the Quantum Fourier Transform is invertible. We call this the Inverse Quantum Fourier Transform (IQFT). The IQFT returns the integer x from our QFT equation encoded in binary by an m-qubit state (Kaye et al., 2007, p. 117). Recall the QFT was written as

$$QFT(|x\rangle) = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} e^{2\pi i x k/N} \, |k\rangle \, .$$

We can write the IQFT as

$$IQFT(|k\rangle) = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} e^{-2\pi i x k/N} |k\rangle .$$

We used this idea in our modified version of Shor's algorithm to see what each IQFT would return. To implement the classical version of IQFT, we used the Inverse Fast Fourier Transform algorithm in MATLAB to invert the absolute value of $g(k)$ as a function in k. For our initial random variable, $a = 7$, a plot of the IFFT is provided. The remaining plots of the IFFT for all random variables $1 \leq a < 33$ are provided in Appendix B. The importance of the being able to sudo-implement the Inverse Quantum Fourier Transform is that we have confirmed the QFT is invertible and we our able to examine the periodic nature of our 2^m qubits in superposition. By understanding the initial waveform of the absolute values of $g(k)$, i.e. the IFFT, we are able to see how the periodicity translates to the absolute value of $g(k)$ in the domain created by applying the QFT.

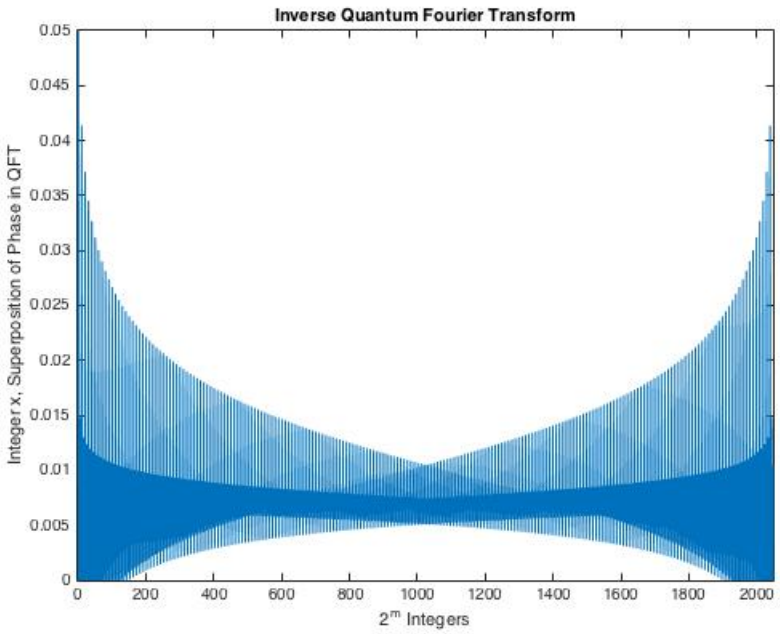


Figure 10:
IFFT outputs x.

COMPLEXITY TIME

An implementation to illustrate the complexity time of our version of Shor's Algorithm using MATLAB version 8.5.0.197613(R2015a) is provided below. We factored several integers with select random variables, which were chosen for no particular reason, to demonstrate the complexity time of our procedure.

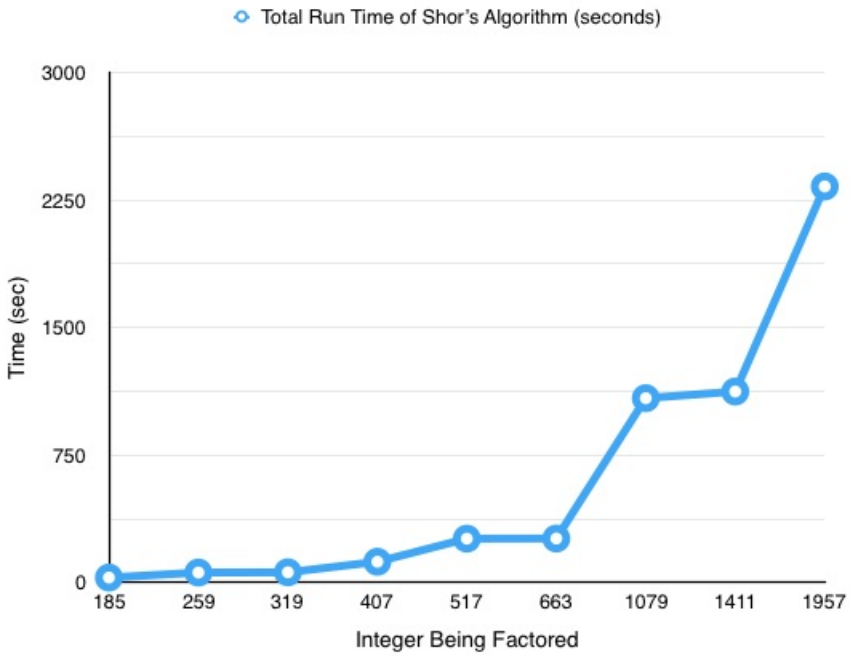


Figure 11:
Total Run Time for Shor's Algorithm

As we increase the size of the integer to factor into a product of two primes, the total run time of Shor's algorithm appears to increase exponentially. This is because we are using a classical computer and do not have the power of massive parallel computing to see a polynomial run time.

CONCLUSION

The theory of quantum computing is a captivating area of research. Here, we presented a general overview of quantum phenomena as well as a fundamental overview of quantum computers. There are numerous books devoted to these topics so our focus was on providing enough detail and information to demystify Shor's algorithm.

We successfully implemented Shor's algorithm, which is intended for use in quantum computers, on a classical computer. In order to do so, some modifications were made to the algorithm. Because classical computers do not have the physical capabilities to employ quantum phenomena, we had to use classical methods to obtain the same results as if this algorithm had been implemented on a quantum computer. One necessary modification made to Shor's algorithm was to apply a Fast Fourier Transform to obtain the Discrete Fourier Transform of our quantum system. From here we could derive the period of some periodic function which would return a factor of n with high probability.

We examined how the random variable in Shor's algorithm affected the entire procedure. After case-by-case analysis for each random variable $1 \leq a < n$, we discovered that there was experimentally a 25% chance of obtaining a correct factor, not including the trivial factors 1 and n. This is unfortunately low considering the overall power and ramifications of Shor's algorithm itself. Shor's algorithm has the capability of breaking the common RSA cryptosystem if quantum computers can ever be made with sufficiently large quantum memory registers.

If we did not obtain a factor of n the first time we implemented Shor's algorithm, based on our experimental results, we would have $1 - \frac{3}{4}^2$ chance of correctly factoring

n after a second iteration of the algorithm. Note that as the number of iterations increases, the probability of actually obtaining a correct factor of n drastically increases. Hence, it would not take many iterations before we finally obtain a factor. In our implementation of factoring 33, the probability of successfully obtaining some factor of 33 after 3 runs would put us at a 57% chance of computing a factor. In worst case scenario, 16 implementations of Shor's algorithm would give us a 99% chance of actually finding a factor of 33. We would not have to run through all possible 32 random variables in order to find some factor.

Even though there is no direct correlation between the random variable selected and the success of returning a non-trivial factor of n, we were able to demonstrate exactly where Shor's algorithm would fail. This gave us more insight into the complexity of this algorithm and showed us that even though the algorithm is quite efficient, there is still room for improvement. One modification I would like to see is figuring out some way to increase the percentage likelihood of obtaining a non-trivial factor of n so that the number of iterations of Shor's algorithm itself would be minimized. In other words, by utilizing another mathematical method, would we be able to find a non-trivial factor of n no matter which random variable we've selected?

Our modified version of Shor's algorithm was able to run in exponential time. As the size of integers we wish to factor increases, the complexity time would drastically increase. Eventually, it would be necessary to exploit quantum phenomena in order to successfully find non-trivial factors of some integer.

I have tried to illustrate that although Shor's algorithm is efficient, the probability of actually obtaining some factor of n immediately remains low. I firmly believe

that using probabilistic and statistic-based methods to better define which random variable we select and why would significantly increase our chances of finding some factor immediately rather than having to iterate the algorithm a possible $n - 1$ number of times. This would reduce the computational time of the algorithm which in turn would make Shor's algorithm even more powerful. And isn't this the goal anyway? Who can solve the prime factorization problem and break RSA the fastest?

MATLAB CODE USED IN THESIS

Shor's Algorithm using MATLAB

```
% IMPLEMENTATION OF SHOR'S ALGORITHM
% March 26, 2016
% Author: Elizabeth Parsons

% Composite Integer to factor
n = 33;
% Compute n^2 <= 2^m < 2n^2 and find m
for i = 1:n/2+1
    if n^2 <= 2^i && 2^i <2*n^2;
        m = i
    end
end
% Select random variable a with 1 < a < n
a = 7;
% Compute f(x) = a^x mod n
for x = 1:2^m
    f(x) = bigmod(a,x,n);
end
% multiply f by factor
```

```
20 newf = 1/sqrt(2^m)*f;
21 % Compute the period
22 p = seqperiod(f);
23 % find all kets with output 2
24 output2 = find(f==7);
25 % Compute g(c)
26 for c= 1:2^m
27     g(c) = exp(2*sqrt(-1)*pi*c*5/2^m);
28 end
29 % Take absolute value of g(c) times factor
30 g = abs(g)*1/sqrt(2^m);
31 % Compute FFT
32 letbirdy = (1/sqrt(2^m))*abs(fft(newf));
33 % Compute IFFT
34 pigeon = ifft(letbirdy);
35 % Plot IFFT
36 figure;
37 plot(pigeon)
38 axis([0,2^m,0,0.05])
39 % Plot IFFT
40 figure;
41 plot(pigeon)
42 axis([0,2^m,0,5])
```

```
43 % Plot FFT
44 figure;
45 plot((1/sqrt(2^m))*abs(fft(newf)))
46 axis([0,2^m,0,0.2])
47 figure;
48 stem((1/sqrt(2^m))*abs(fft(newf)))
49 axis([0,2^m,0,5])
50 % Normalize the Probability
51 for w = 1:2^m
52     probsum(w) = letbirdy(w)^2;
53 end
54 totalsum = sum(probsum);
55 %peak = findpeaks(f,x,'MinPeakProminence',4,'Annotate','
       extents')
56 [peak, loc] = findpeaks(letbirdy);
57 % pick one of the locs
58 % pk = loc(4);
59 % Compute the number of peaks and select the last peak to
       use
60 lenloc = length(loc);
61 pkloc = lenloc(end);
62 pk = loc(pkloc)
63 if lenloc == 0
```

```
64      %print('No peaks found or only peak is at 0')
65  end
66  newprob = letbirdy(pk)^2/totalsum;
67  peakprob = 4*newprob;
68  % Apply Method of Continued Fractions to find the period
69  xx(1) = pk/2^m;
70  % Find the Convergent Values
71  aa(3) = floor(xx(1));
72  aa(2) = 0;
73  aa(1) = 0;
74  %format rat
75  for i = 1:n
76      xx(i+1) = 1/(xx(i) - aa(i+2));
77      aa(i+3) = floor(xx(i+1));
78  end
79  % Calculate the Continued Fractions
80  % Initial Parameters
81  p(1) = 0;
82  p(2) = 1;
83  q(1) = 1;
84  q(2) = 0;
85  % Find the period r
86  for nn = 1:length(aa)-2
```

```
87      p(nn+2) = aa(nn+2)*p(nn+1) + p(nn);
88      q(nn+2) = aa(nn+2)*q(nn+1) + q(nn);
89      p(nn+2)/q(nn+2);
90        if q(nn+2) < n
91            r = q(nn+2); % Capture the last denominator < prime
92        end
93  end
94  % Apply the Exponent Factorization Method
95  fac = factor(r);
96  % check that a^r = 1 mod n :)
97  check = mod(a^r,n);
98  % find the number of 2's which are prime factors of the
        period r
99  fspec = find(fac==2);
100 m = fac(end);
101 k = length(fspec);
102 % need to write r as r = 2^k*m where m is some odd integer
103 r = 2^k*m;
104 % Find b(1)
105 b(1) = mod(a^m,n);
106 if mod(b(1),n) == 1
107     %fprintf('Shors algorithm has failed line 147')
108     break % Shor's Algorithm has failed
```

```
109 end
110 % Find a factor of n
111 % Use this if there is not a power of 2 in the period
112 for u = 1:m
113     b(u+1) = mod(b(u)^2,n);
114     if mod(b(u),n) == -1
115         %fprintf('Shors Algorithm has failed 173')
116         break % shor's algorithm has failed
117     end
118     if mod(b(u+1),n) == 1 && mod(b(u),n)~= 1 && mod(b(u),n)~=
            -1
119         gcd(b(u)-1,n)
120     end
121 end
```

Modulo Function for Large Integers

```
1 function remainder = bigmod (number, power, modulo)
2 % modulo function for large numbers, -> number^power(mod
      modulo)
3 % by bennyboss / 2005-06-24 / Matlab 7
4 % I used algorithm from this webpage:
5 % http://www.disappearing-inc.com/ciphers/rsa.html
6
```

```
7  % binary decomposition
8  binary(1,1) = 1;
9  col = 2;
10 while ( binary(1, col-1) <= power-binary(1, col-1) )
11     binary(1, col) = 2*binary(1, col-1);
12     col = col + 1;
13 end
14 % flip matrix
15 binary = fliplr(binary);
16 % extract binary decomposition from number
17 result = power;
18 cols = length(binary);
19 extracted_binary = zeros(1, cols);
20 index = zeros(1, cols);
21 for ( col=1 : cols )
22     if( result-binary(1, col) > 0 )
23         result = result - binary(1, col);
24         extracted_binary(1, col) = binary(1, col);
25         index(1, col) = col;
26     elseif ( result-binary(1, col) == 0 )
27         extracted_binary(1, col) = binary(1, col);
28         index(1, col) = col;
29         break;
```

```
30      end
31  end
32  % flip matrix
33  binary = fliplr(binary);
34  % doubling the powers by squaring the numbers
35  cols2 = length(extracted_binary);
36  rem_sqr = zeros(1, cols);
37  rem_sqr(1, 1) = mod(number^1, modulo);
38  if ( cols2 > 1 )
39      for ( col=2 : cols)
40          rem_sqr(1, col) = mod(rem_sqr(1, col-1)^2, modulo);
41      end
42  end
43  % flip matrix
44  rem_sqr = fliplr(rem_sqr);
45  % compute reminder
46  index = find(index);
47  remainder = rem_sqr(1, index(1, 1));
48  cols = length(index);
49  for (col=2 : cols)
50      remainder = mod(remainder*rem_sqr(1, index(1, col)),
        modulo);
51  end
```

REFERENCES

Bacon, D. (n.d.). Dirac Notation and Basic Linear Algebra for Quantum Computing. Lecture presented in University of Washington, Seattle, WA. Retrieved March 19, 2016, from http://courses.cs.washington.edu/courses/cse599d/06wi /lecturenotes2.pdf

Chen, C. (2001). Digital signal processing: Spectral computation and filter design. New York: Oxford University Press.

Dattani, N. S., & Bryans, N. (2014, December 1). Quantum factorization of 56153 with only 4 qubits. Retrieved March 15, 2016, from http://arxiv.org/pdf/ 1411.6758v3.pdf

Hardy, G. H., & Wright, E. M. (2008). An introduction to the theory of numbers (Vol. 6). Oxford: Clarendon Press.

Heiberg, J. L., Fitzpatrick, R., & E. (2008). Euclid's elements of geometry: The Greek text of J.L. Heiberg (1883-1885): From Euclidis Elementa, edidit et Latine interpretatus est I.L. Heiberg, in aedibus B.G. Teubneri, 1883-1885. Place of publication not identified: Publisher not identified.

Jha, S., Chatterjee, P., Falor, A., & Chakraborty, M. (n.d.). A Matlab Realization of Shor's Quantum Factoring Algorithm. Retrieved March 19, 2016, from https://researchpapers4scolars.files.wordpress.com /2015/06/spsitm2011_submission _55-quantum.pdf

Kaye, P., Laflamme, R., & Mosca, M. (2007). An introduction to quantum computing. Oxford: Oxford University Press.

Kendon, V. M., & Munro, W. J. (2005). Entanglement and its Role in Shor's

Algorithm. Hewlett-Packard Development Company.

Mollin, R. A. (2001). An Introduction to Cryptography. Boca Raton: Chapman & Hall/CRC.

Nielsen, M. A., & Chuang, I. L. (2000). Quantum Computation and Quantum Information (10th ed.). Cambridge: Cambridge University Press.

Riley, K., Hobson, M., & Bence, S. (2006). Mathematical Methods for Physics and Engineering (Vol. 3). Cambridge: Cambridge University Press.

Rivest, R. L., Shamir, A., & Adleman, L. (1977, April 4). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Retrieved March 03, 2016, from https://people.csail.mit.edu/rivest/Rsapaper.pdf

RSA Laboratories. (n.d.). Retrieved March 25, 2016, from http://www.rsasecurity .com.tw/emc-plus/rsa-labs/historical/a-cost-based -security-analysis -key-lengths.htm

Shannon Number. (2016, March 16). Retrieved March 22, 2016, from https://en.wikipedia.org/wiki/Shannon_number

Shor, P. W. (1994). Algorithms for Quantum Computation: Discrete Logarithms and Factoring. Retrieved February 03, 2016, from http://www.csee.wvu.edu/ xinl/library/papers/comp/shor_focs1994.pdf

Shor, P. W. (1997). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM Journal on Computing, 26(5), 1484-1509. Retrieved February 05, 2016.

Smith, S. W. (1997). The Scientist and Engineer's Guide to Digital Signal Processing. San Diego, CA: California Technical Publishing.

Terr, David. "Polynomial Time." From MathWorld–A Wolfram Web Resource,

created by Eric W. Weisstein. http://mathworld.wolfram.com/ Polynomial-Time.html

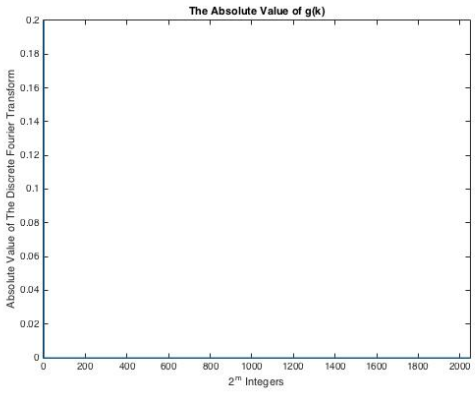
Trappe, W., & Washington, L. C. (2006). Introduction to cryptography: With coding theory (2nd ed.). Upper Saddle River, NJ: Pearson Prentice Hall.

Vandersypen, L. M., Steffen, M., Breyta, G., Yannoni, C. S., Sherwood, M. H., & Chuang, I. L. (2001). Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. Nature, 414(6866), 883-887. Retrieved March 22, 2016.

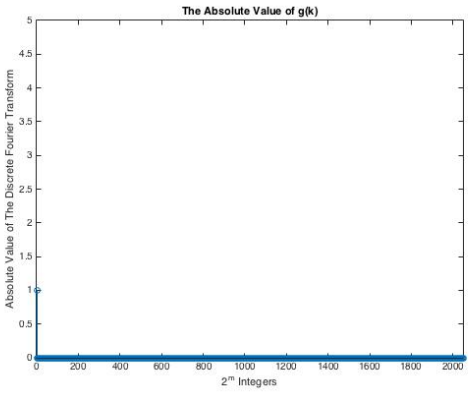
Vries, A. D. (2012). Quantum Computation: An Introduction for Engineers and Computer Scientists. Norderstedt: Books on Demand.

APPENDIX A

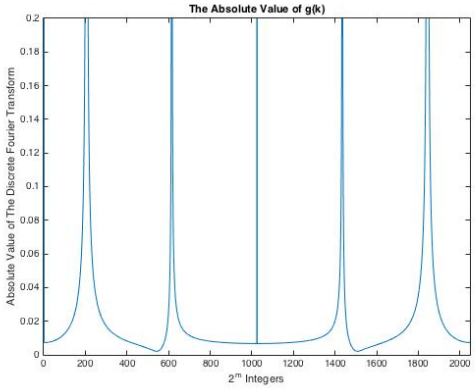
This appendix provides all plots of the absolute value of $g(k)$ for random variables $1 \leq a < 33$ with a measurement made at 7.



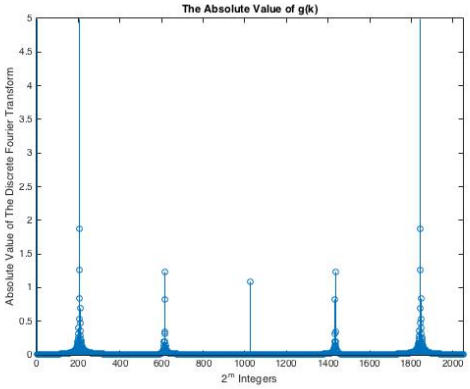
(a) Random Variable = 1



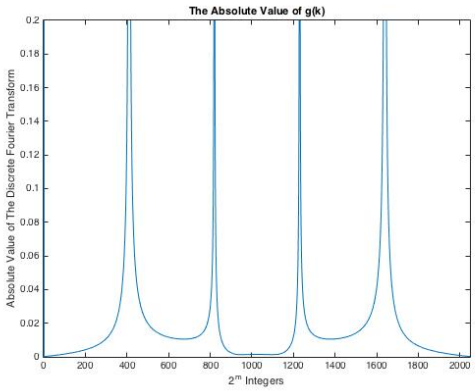
(b) Random Variable = 1



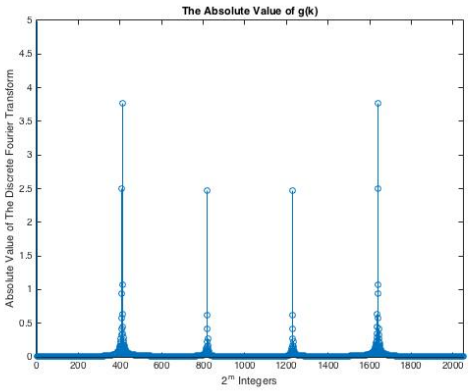
(c) Random Variable = 2



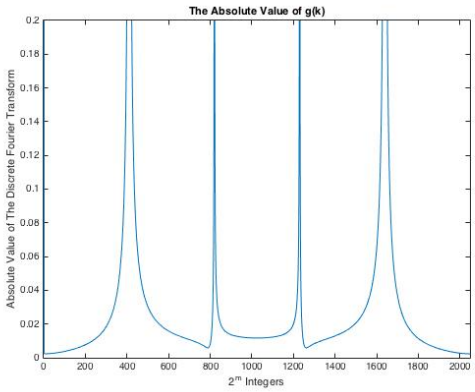
(d) Random Variable = 2



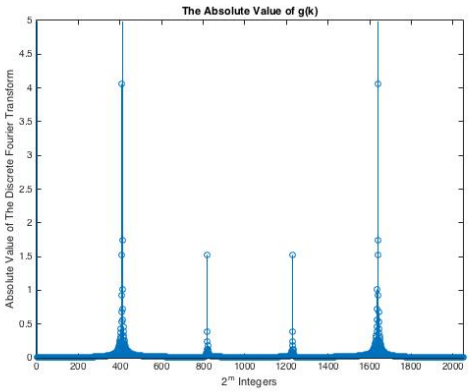
(a) Random Variable = 3



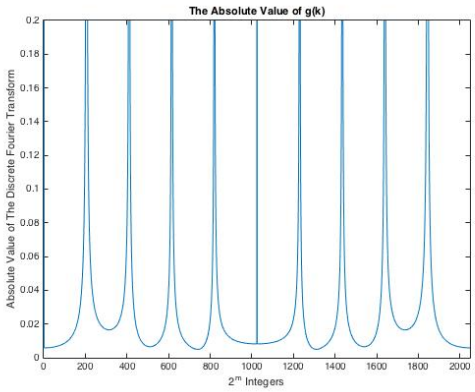
(b) Random Variable = 3



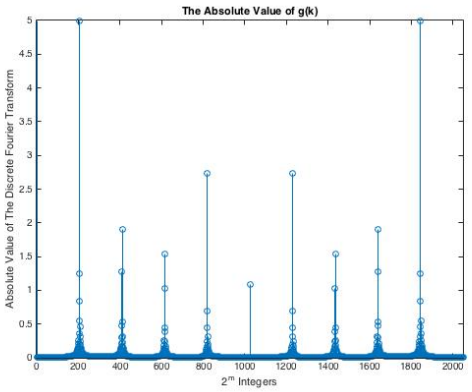
(c) Random Variable = 4



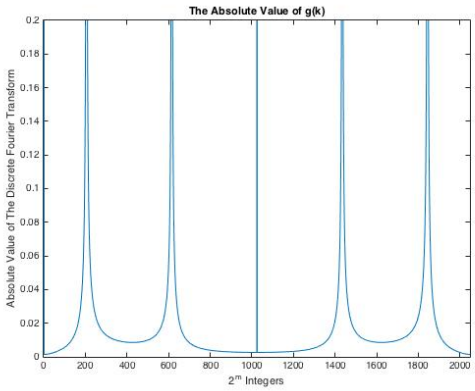
(d) Random Variable = 4



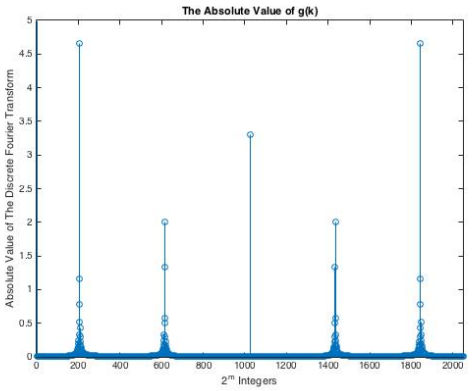
(e) Random Variable = 5



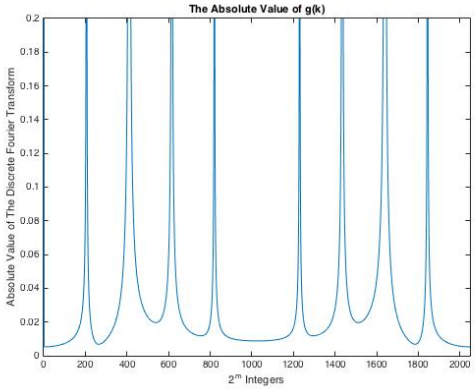
(f) Random Variable = 5



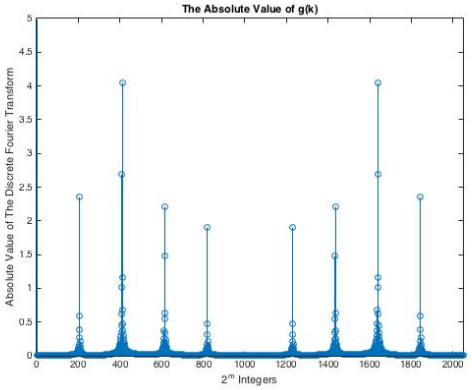
(a) Random Variable = 6



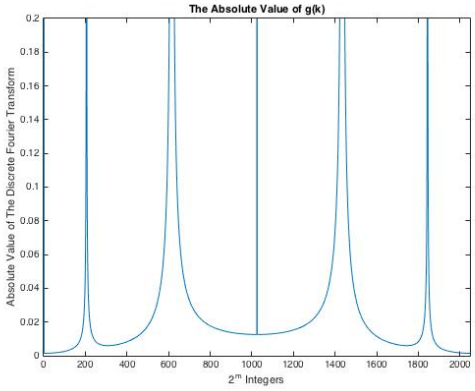
(b) Random Variable = 6



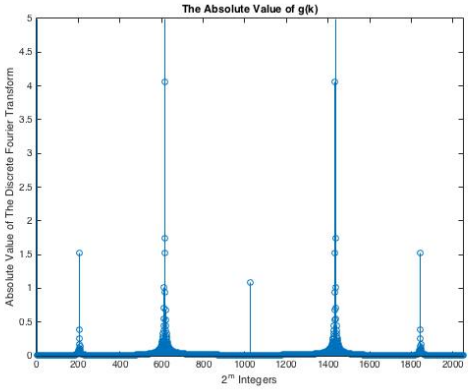
(c) Random Variable = 7



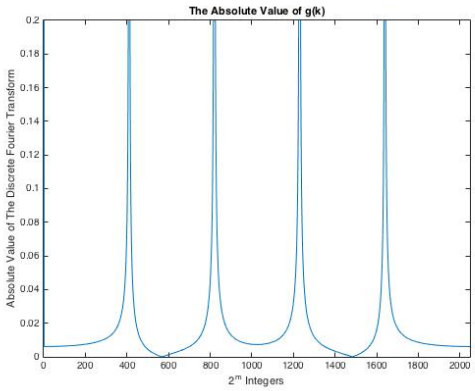
(d) Random Variable = 7



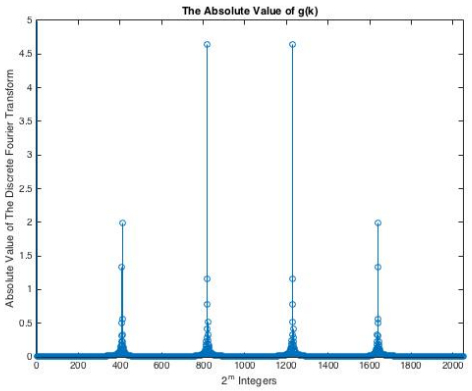
(e) Random Variable = 8



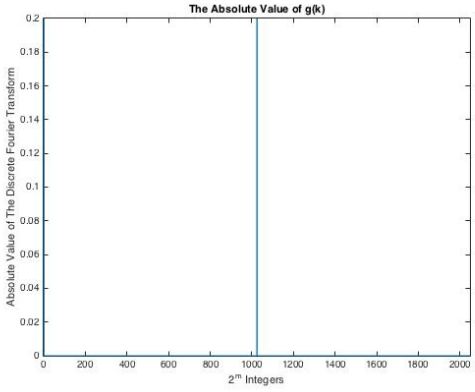
(f) Random Variable = 8



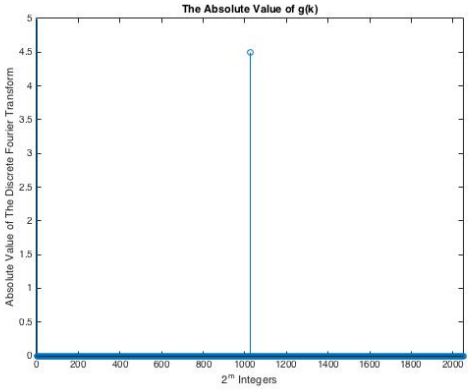
(a) Random Variable = 9



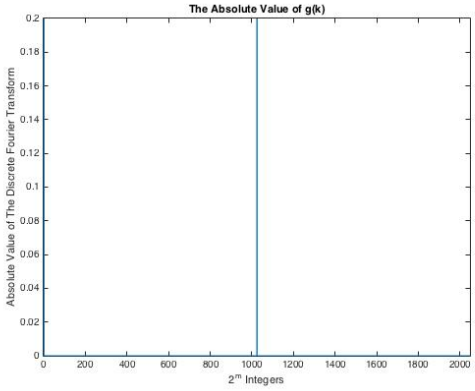
(b) Random Variable = 9



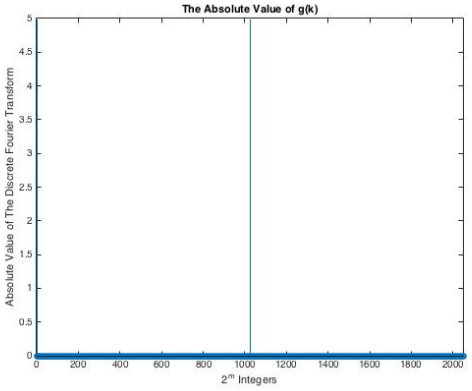
(c) Random Variable = 10



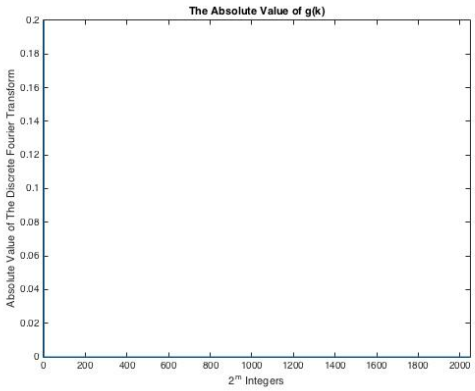
(d) Random Variable = 10



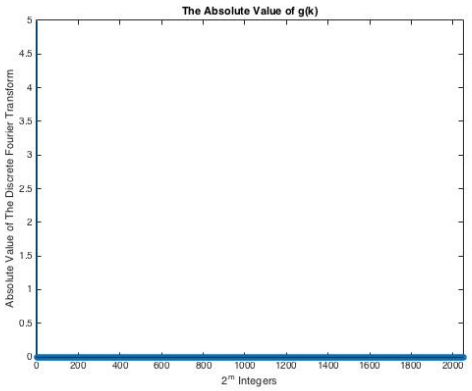
(e) Random Variable = 11



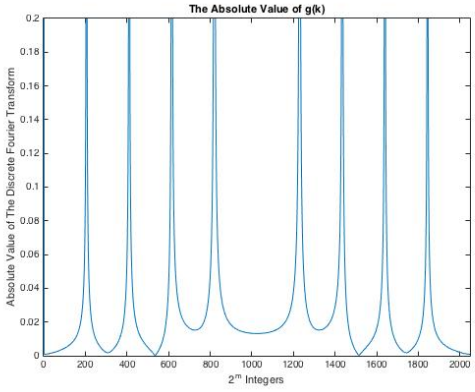
(f) Random Variable = 11



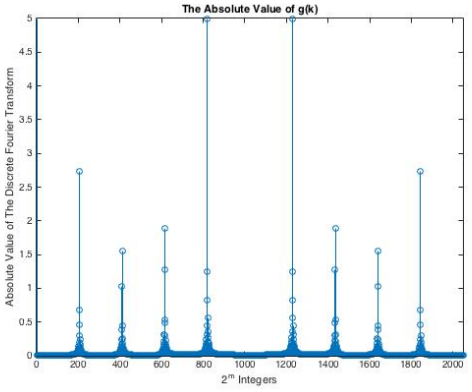
(a) Random Variable = 12



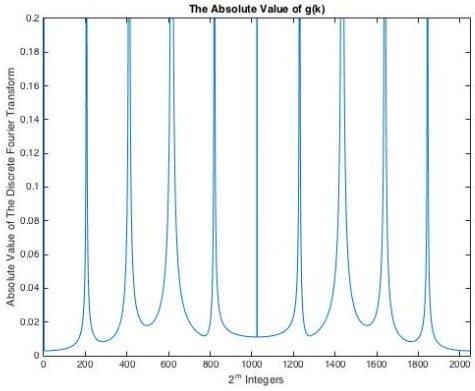
(b) Random Variable = 12



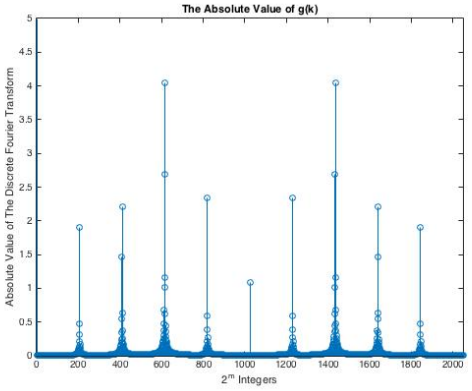
(c) Random Variable = 13



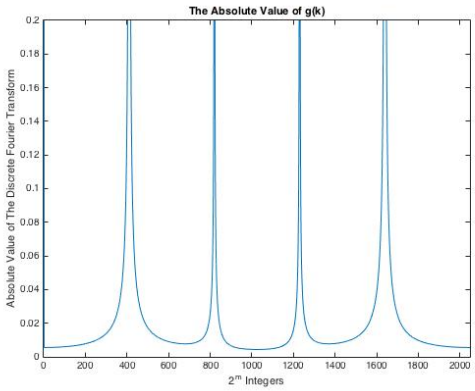
(d) Random Variable = 13



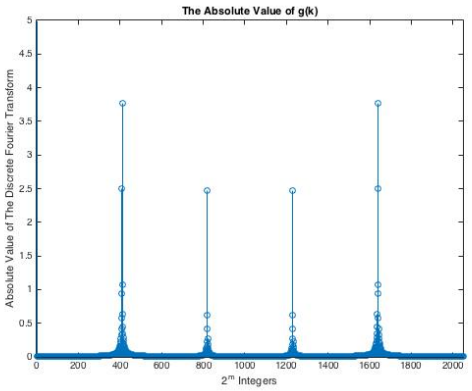
(e) Random Variable = 14



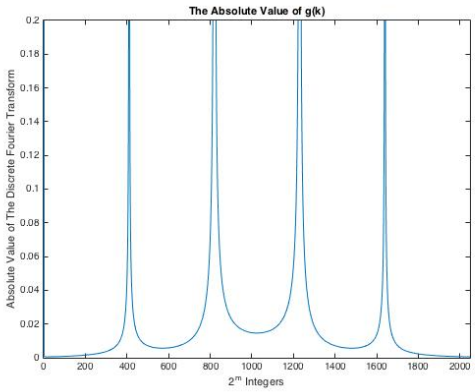
(f) Random Variable = 14



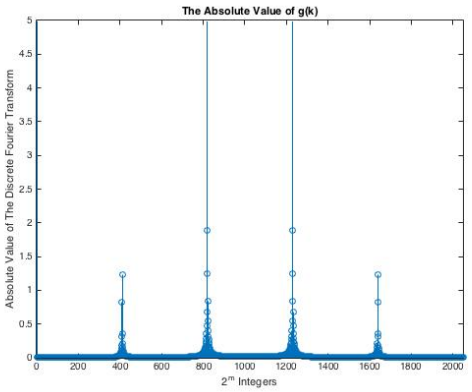
(a) Random Variable = 15



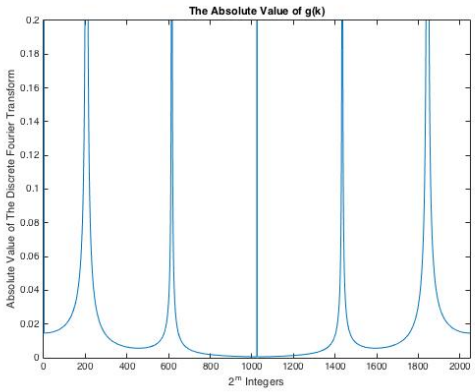
(b) Random Variable = 15



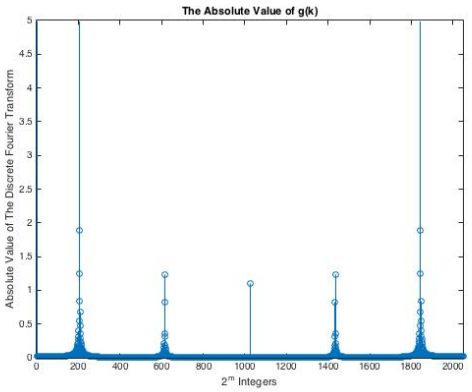
(c) Random Variable = 16



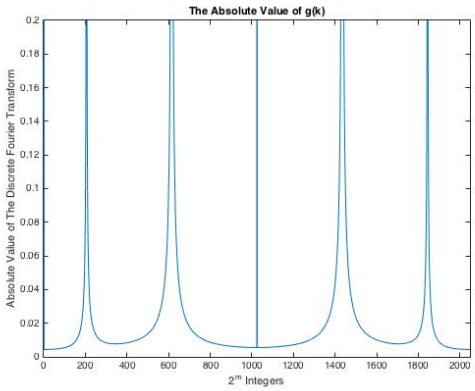
(d) Random Variable = 16



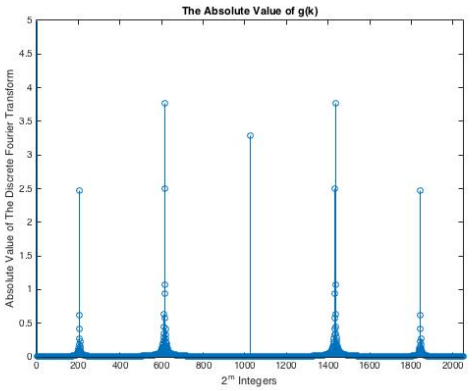
(e) Random Variable = 17



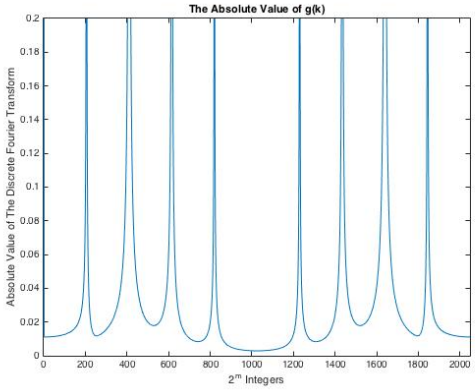
(f) Random Variable = 17



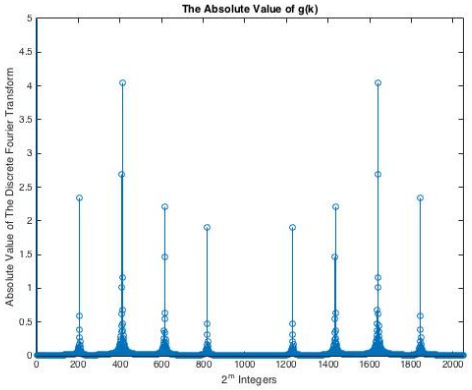
(a) Random Variable = 18



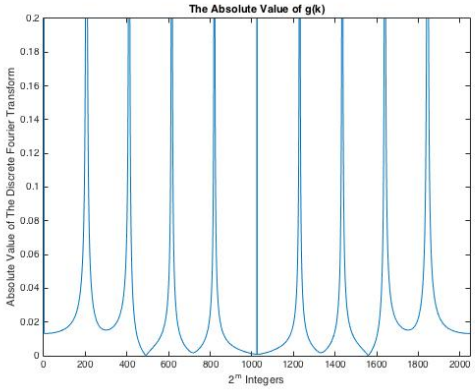
(b) Random Variable = 18



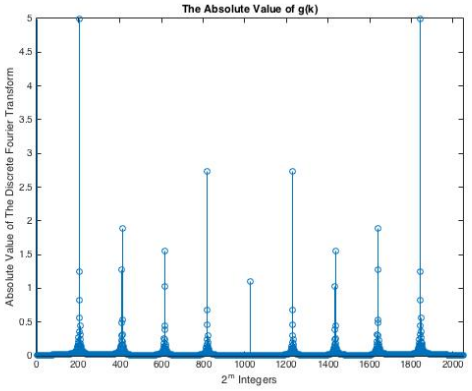
(c) Random Variable = 19



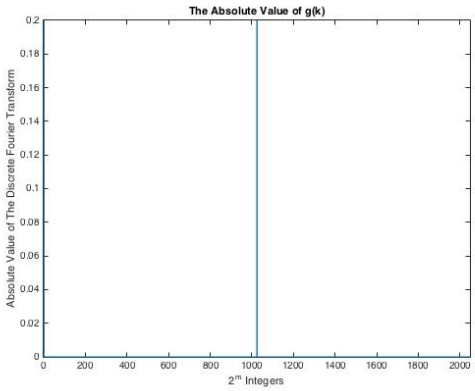
(d) Random Variable = 19



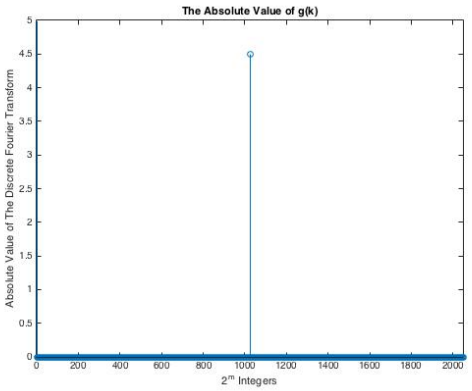
(e) Random Variable = 20



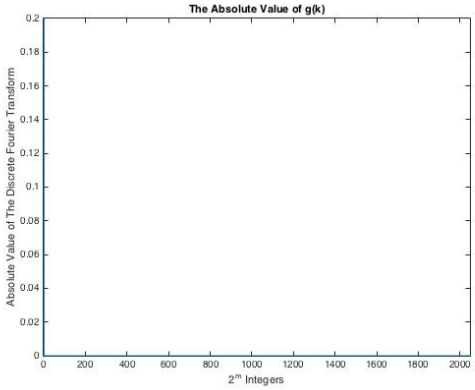
(f) Random Variable = 20



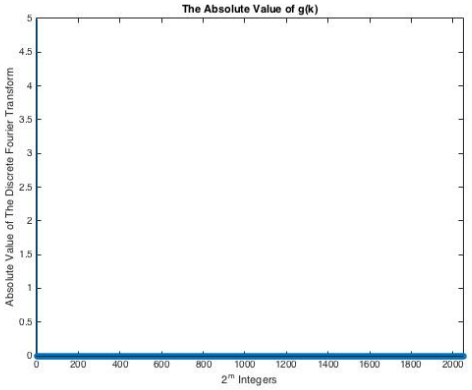
(a) Random Variable = 21



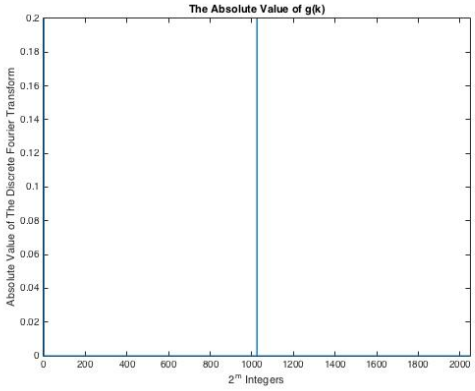
(b) Random Variable = 21



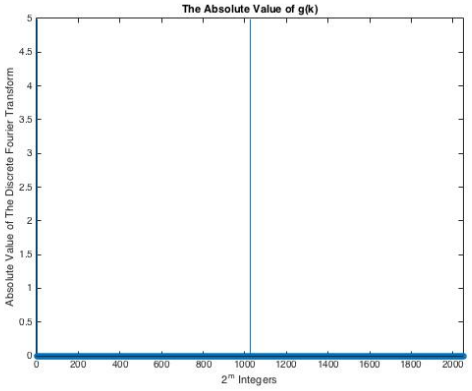
(c) Random Variable = 22



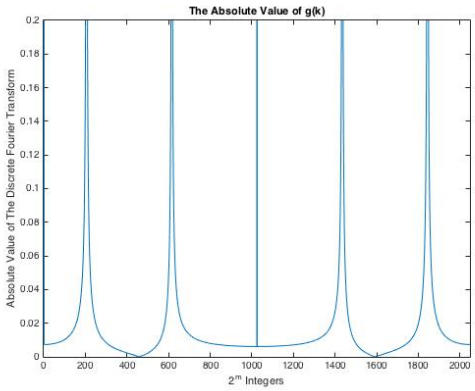
(d) Random Variable = 22



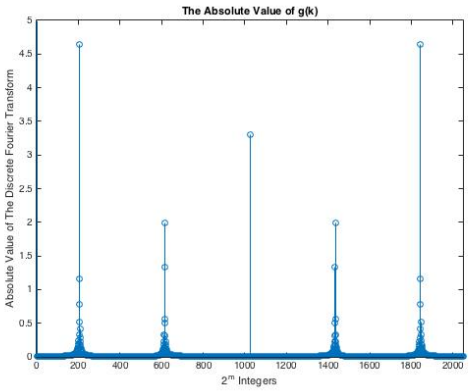
(e) Random Variable = 23



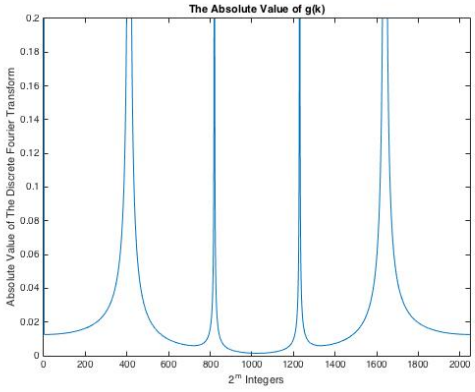
(f) Random Variable = 23



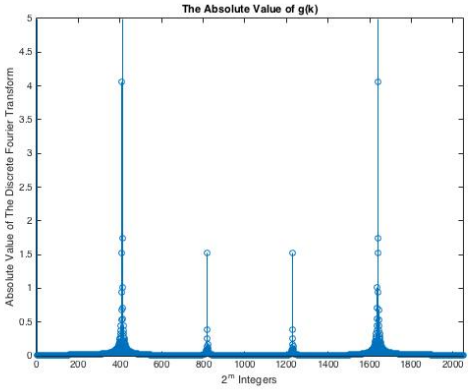
(a) Random Variable = 24



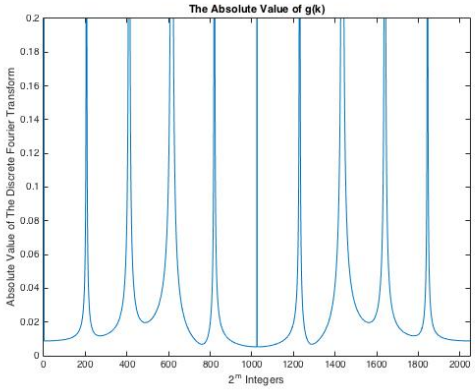
(b) Random Variable = 24



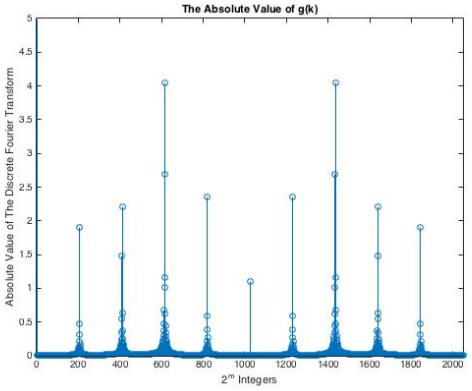
(c) Random Variable = 25



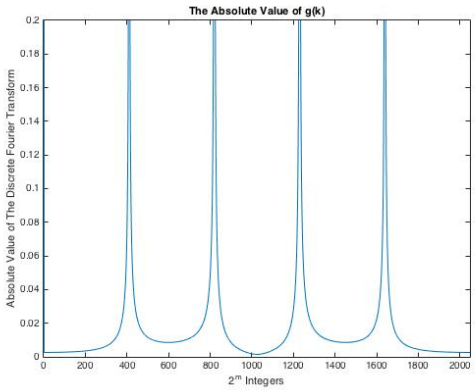
(d) Random Variable = 25



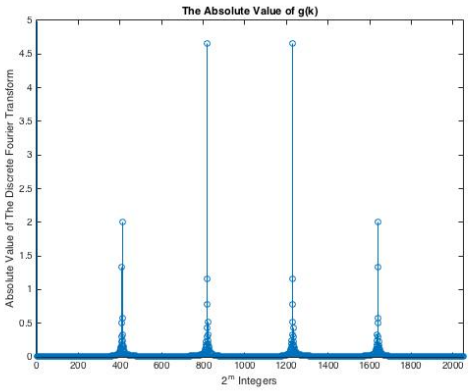
(e) Random Variable = 26



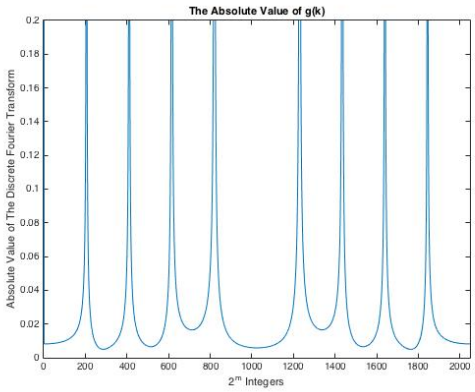
(f) Random Variable = 26



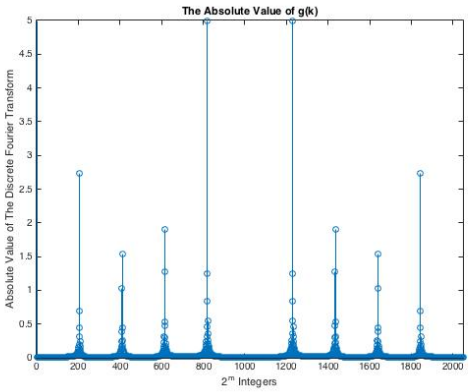
(a) Random Variable = 27



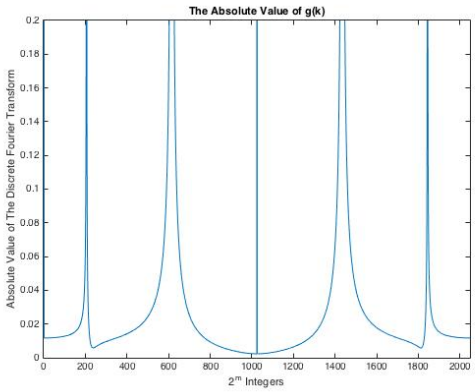
(b) Random Variable = 27



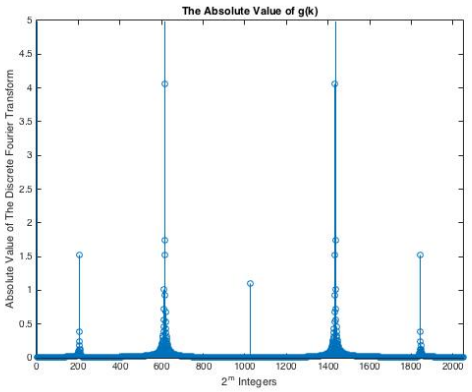
(c) Random Variable = 28



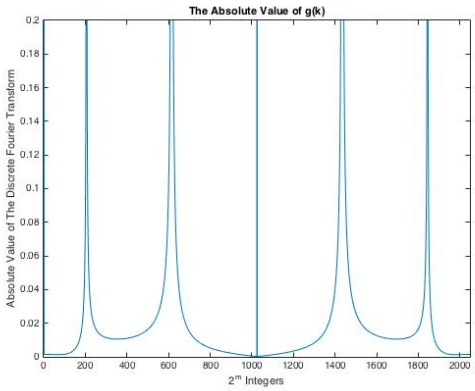
(d) Random Variable = 28



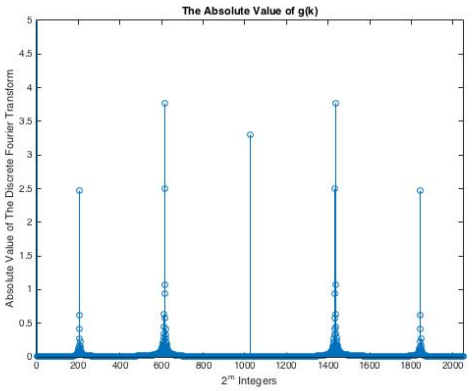
(e) Random Variable = 29



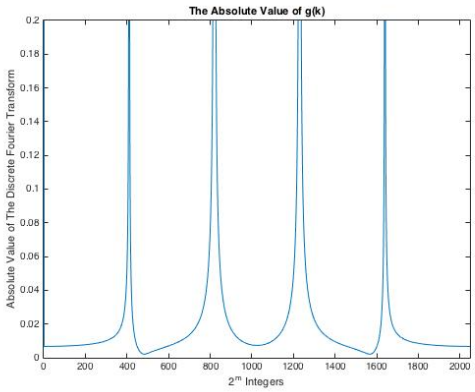
(f) Random Variable = 29



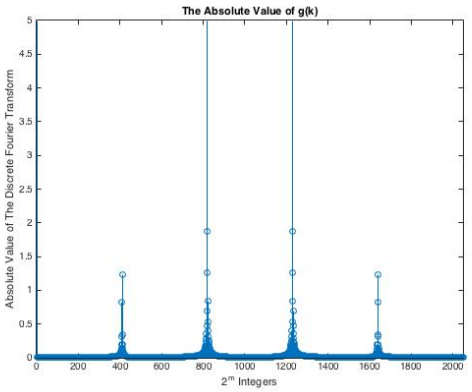
(a) Random Variable = 30



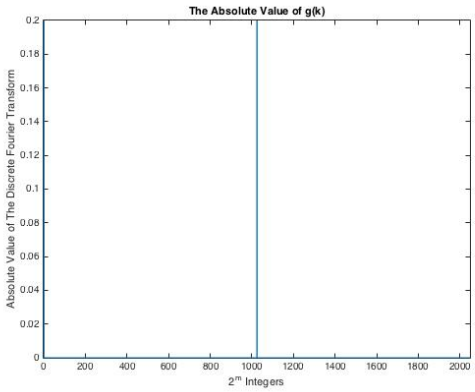
(b) Random Variable = 30



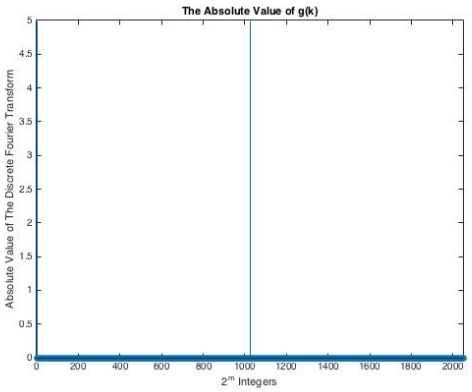
(c) Random Variable = 31



(d) Random Variable = 31



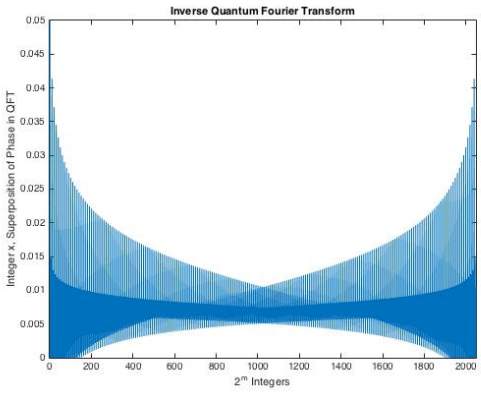
(e) Random Variable = 32



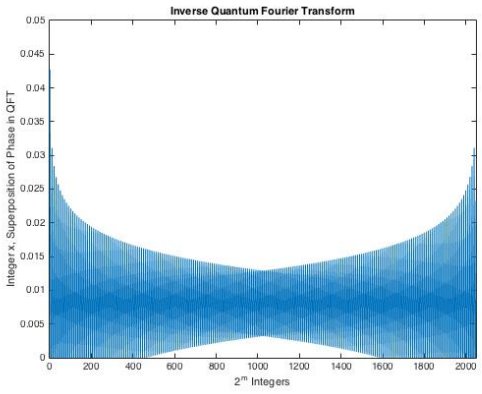
(f) Random Variable = 32

APPENDIX B

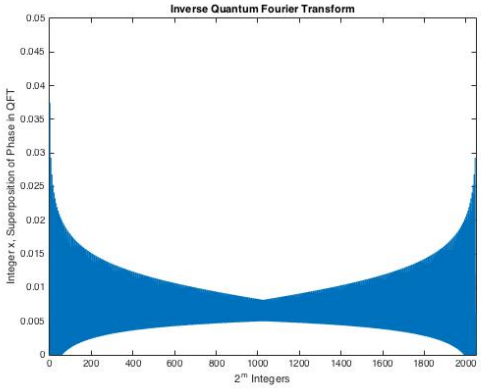
This appendix provides all plots of the Inverse Fast Fourier Transform for random variables $1 \leq a < 33$ with a measurement made at 7.



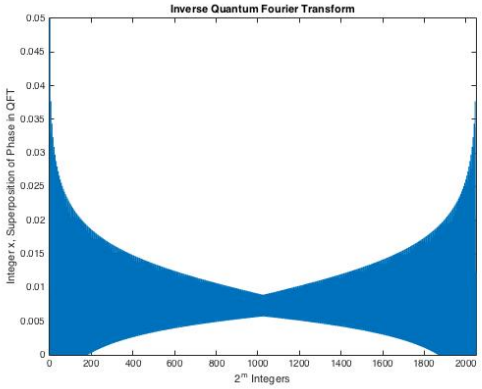
(a) Random Variable = 1



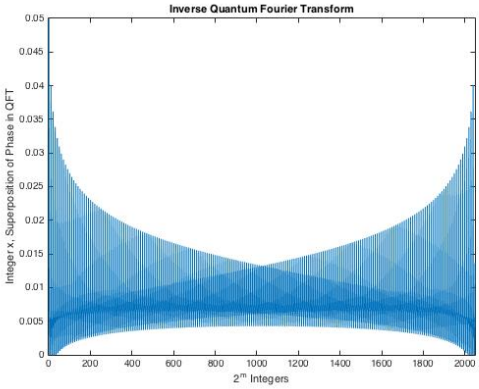
(b) Random Variable = 2



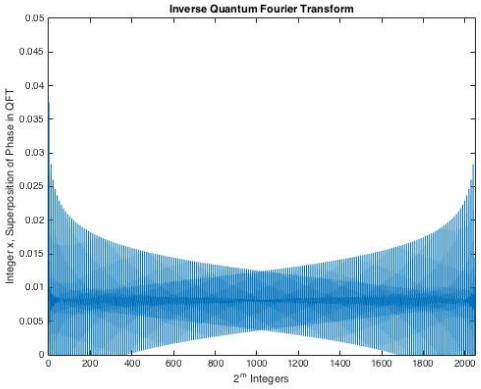
(c) Random Variable = 3



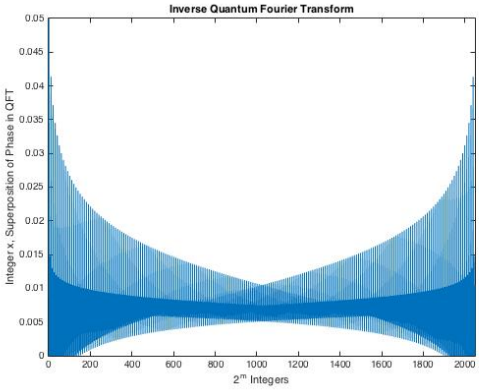
(d) Random Variable = 4



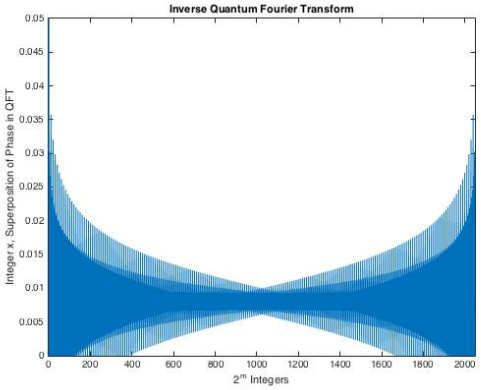
(a) Random Variable = 5



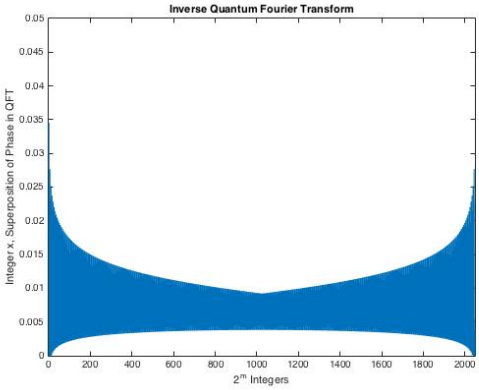
(b) Random Variable = 6



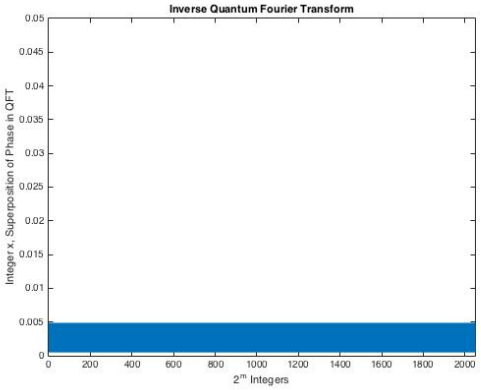
(c) Random Variable = 7



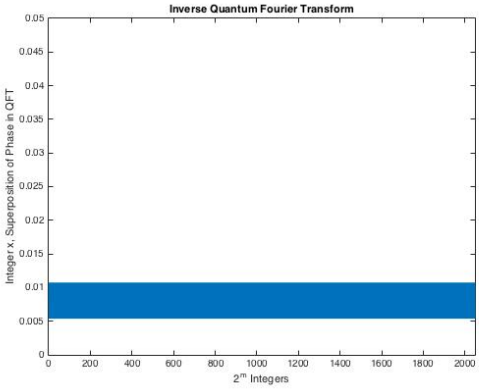
(d) Random Variable = 8



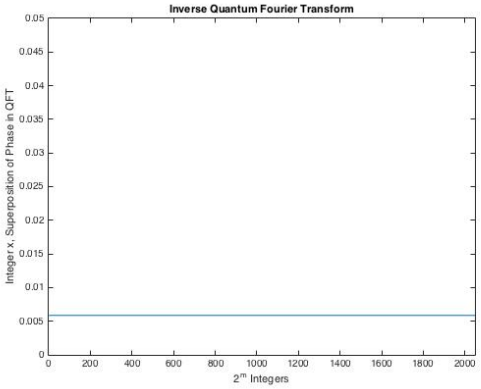
(e) Random Variable = 9



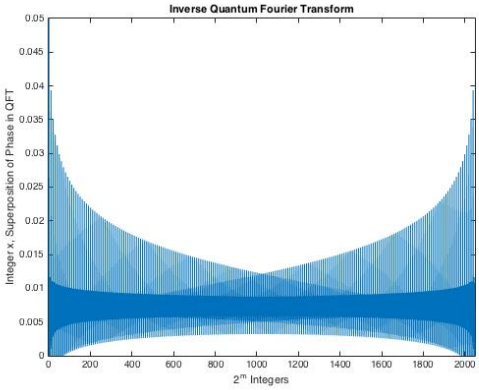
(f) Random Variable = 10



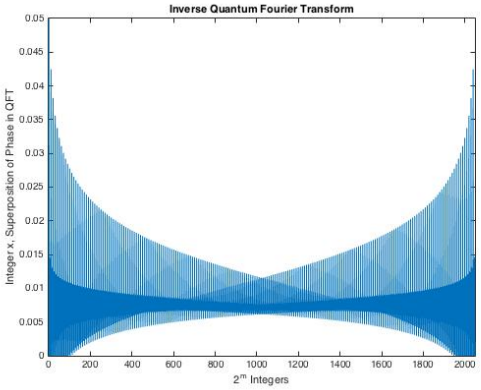
(a) Random Variable = 11



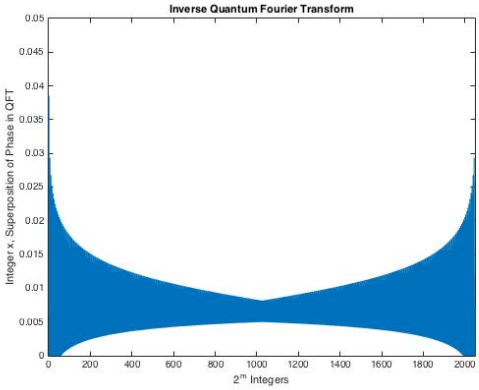
(b) Random Variable = 12



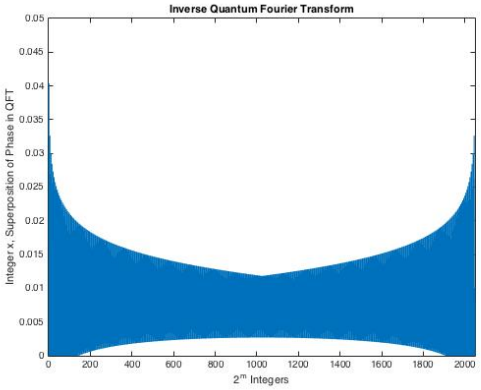
(c) Random Variable = 13



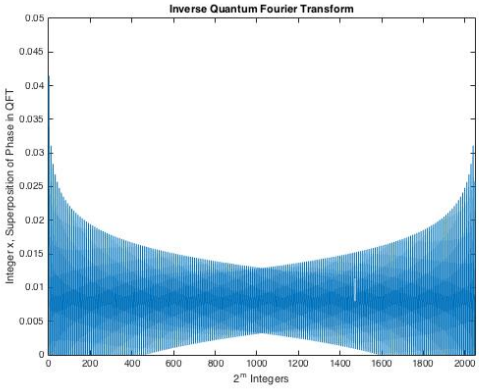
(d) Random Variable = 14



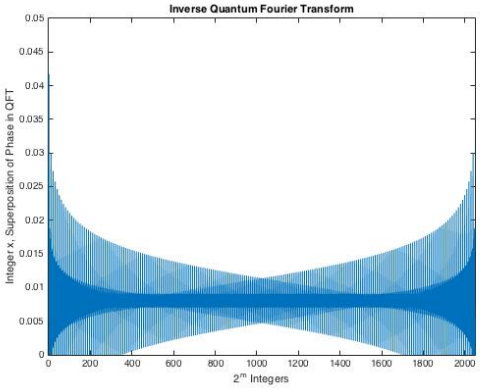
(e) Random Variable = 15



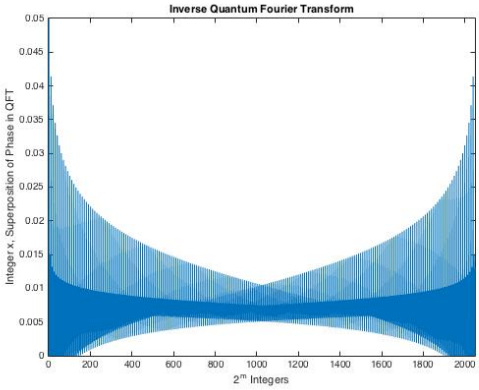
(f) Random Variable = 16



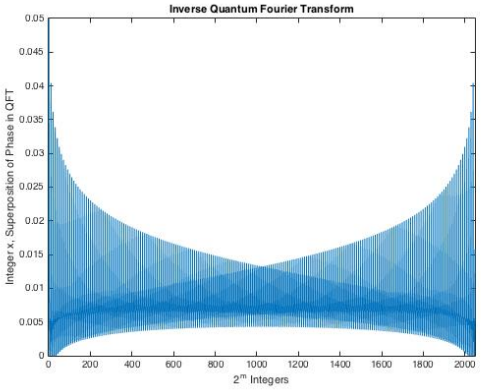
(a) Random Variable = 17



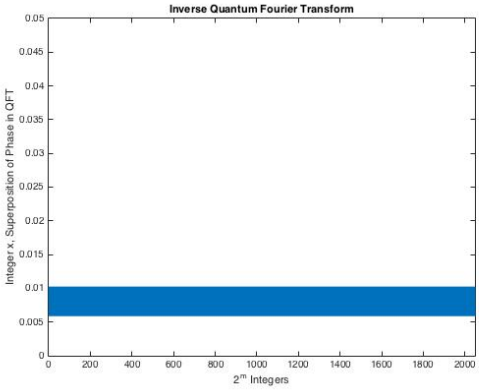
(b) Random Variable = 18



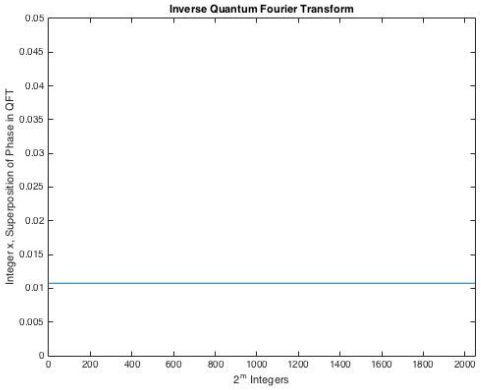
(c) Random Variable = 19



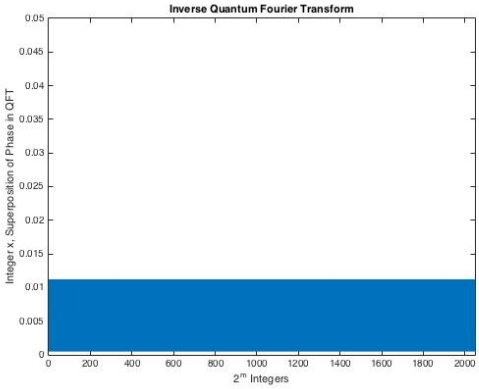
(d) Random Variable = 20



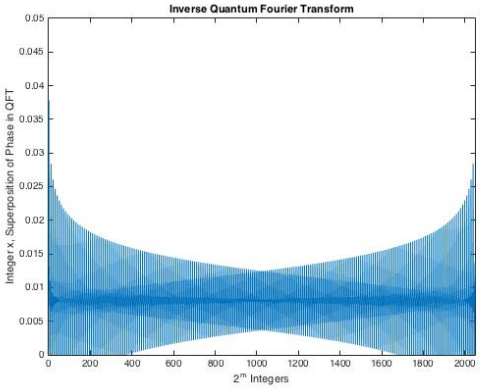
(e) Random Variable = 21



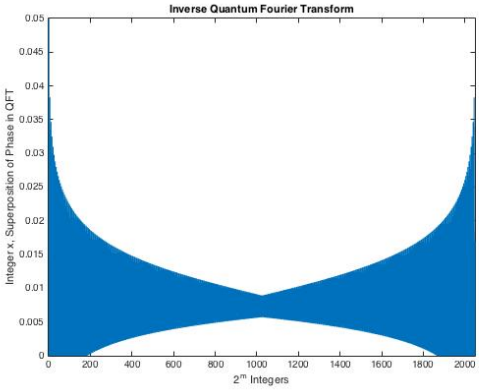
(f) Random Variable = 22



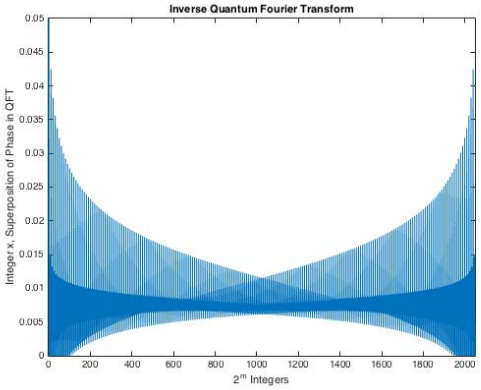
(a) Random Variable = 23



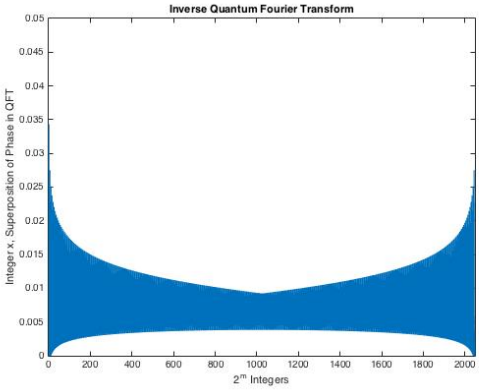
(b) Random Variable = 124



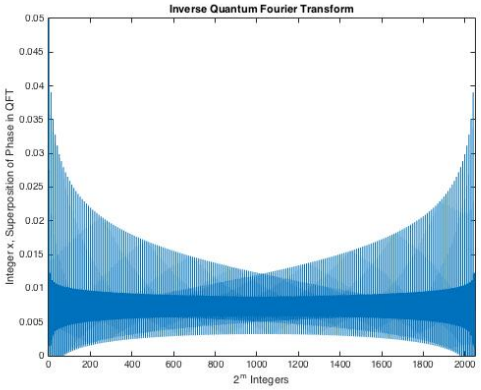
(c) Random Variable = 25



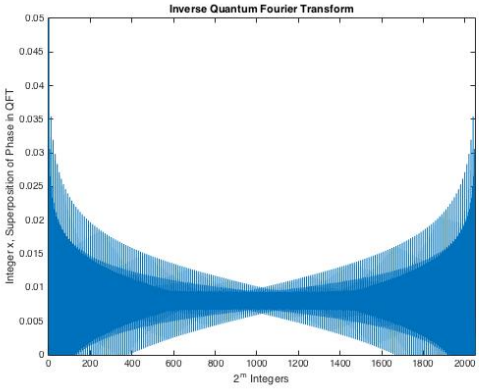
(d) Random Variable = 26



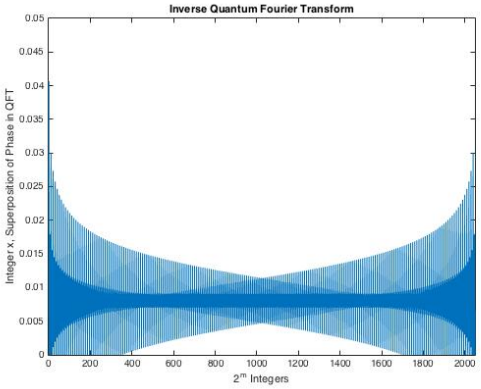
(e) Random Variable = 27



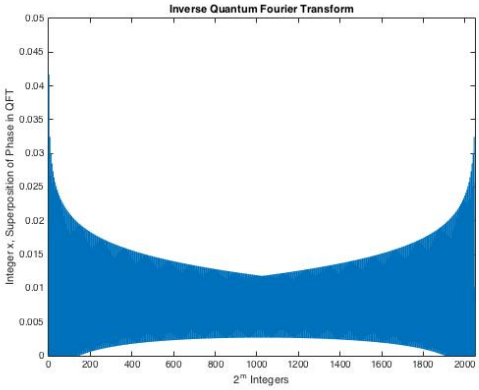
(f) Random Variable = 28



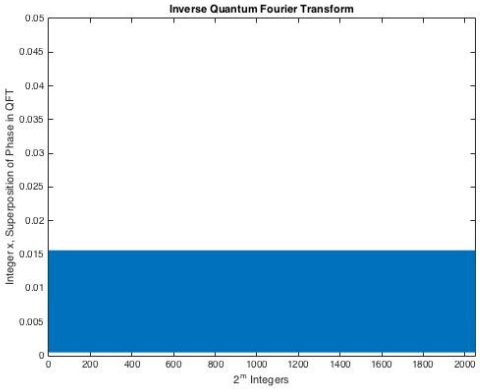
(a) Random Variable = 29



(b) Random Variable = 30



(c) Random Variable = 31



(d) Random Variable = 32