**Quantum Computing with MATLAB**

**By-**

**Rohan Bhatia**

**rohanbhatia\_2k19ep079@dtu.ac.in**

**CONTENTS**

* **Introduction**
* **Qubit**
* **Dirac Notation and Hilbert Space**
* **Quantum Register**
* **Quantum Entanglement**
* **Quantum Fourier Transform**
* **Hadamard Gate and Transformation**
* **Introduction Shor’s Algorithm**
* **Shor’s Algorithm and MATLAB code**
* **Conclusion**

**INTRODUCTION**

The intent of this project is to elucidate the quantum computing algorithms; Shor’s algorithm and Grover’s Algorithm. We will provide a detailed description and simulation of these algorithms 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 both algorithms.

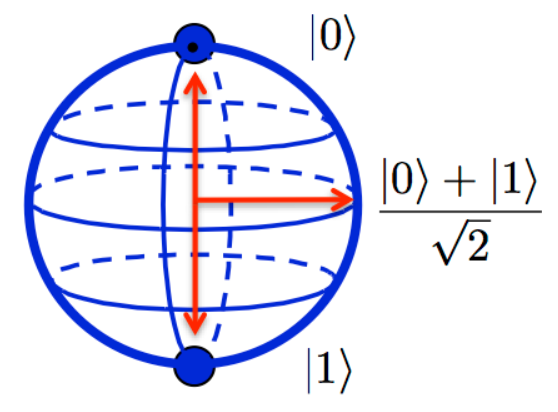
Quantum computers store and transport information quite differently than their classical counterparts. We will provide a quick overview of these differences to highlight the benefit of utilizing quantum phenomena in a computer in order to create massive parallel computations. Thus, reducing the complexity 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.

**QUBITS**

Qubit is the basic unit of quantum information. In classical computing the information is encoded in bits, where each bit can have the value zero or one. In quantum computing the information is encoded in qubits. 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,

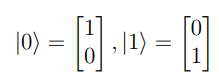


Qubits are only in a state of superposition before any measurement has been made. Once a measurement is made on the quantum state, the superposed state of a qubit collapses to |0> or |1>. We never actually see that a qubit is in a superposition because by simply measuring the state, we cause an effect which collapses the state.

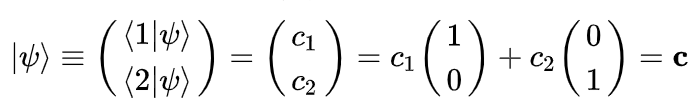
Supposing the two available basis states of the system are |1> and |2>, then in general the state can be written as a superposition of these two states with probability amplitudes c1 and c2

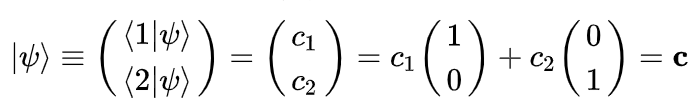


The contents of Dirac bras and kets are labels that describe the underlying vectors. |0> and |1> may be transformed into any two vectors that form an orthonormal basis in C2. The most common basis used in quantum computing is called the computational basis:

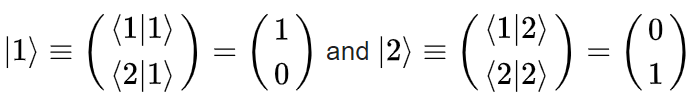


But any other orthonormal basis could be

These two complex numbers may be considered coordinates in a two-dimensional complex Hilbert space. Thus the state vector corresponding to the state  is-

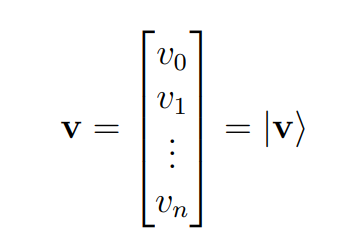


And the basis states correspond to the basis vectors,

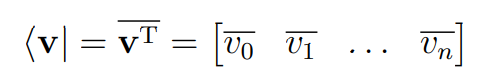


**Dirac notation and Hilbert Space**

In Dirac notation:



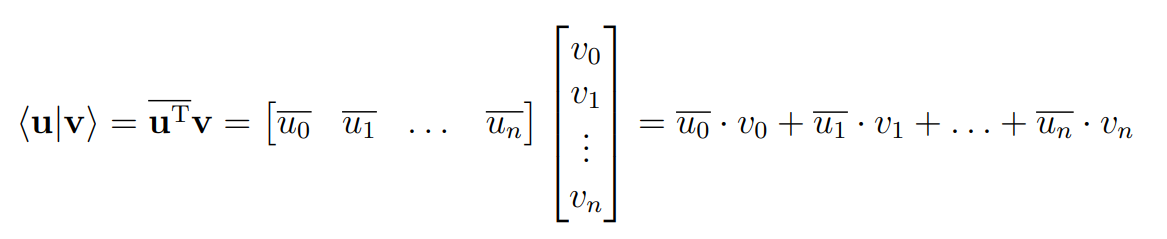
This column vector is referred to as “ket-v.” The dual vector of ket-v has the following Dirac notation:



Where v is the complex conjugate of v. This dual vector is called “bra-v.”

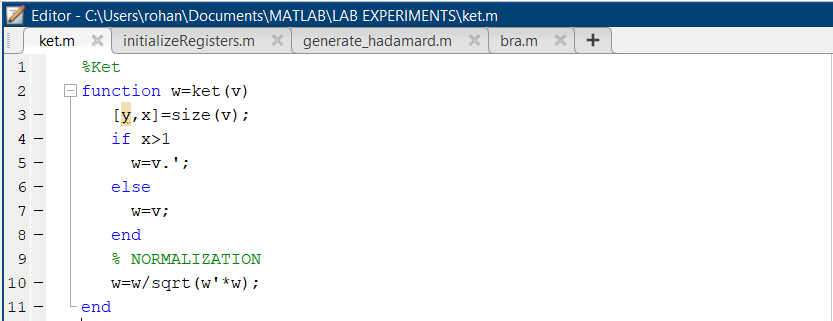
In quantum mechanics the state of a physical system is represented by a vector in a Hilbert space: a complex vector space with an inner product. The term “Hilbert space” is often reserved for an infinite-dimensional inner product space having the property that it is complete or closed. Hilbert spaces serve to clarify and generalize the concept of Fourier expansion and certain linear transformations such as the Fourier transform.

Dirac notation is a convenient way to describe vectors in the Hilbert space Cn, which is the vector space that is most useful for reasoning about quantum systems. A Hilbert space is a vector space with an inner product, and a norm defined by that inner product. The inner product of a vector space is an operation that assigns a scalar value to each pair of vectors u and v in the vector space, and the inner product of two vectors in a Hilbert space Cn is denoted using the Dirac notation <u|v>. Thus the inner product <u|v> of two vectors in a complex Hilbert space is given by the dot product of the vectors v and uT, the conjugate transpose of u:



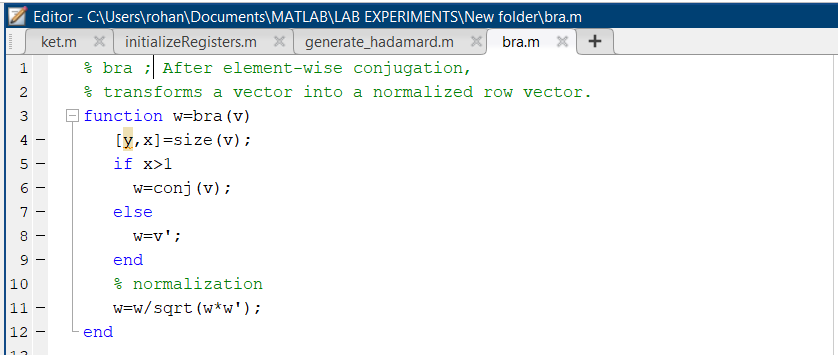
**Implementation in MATLAB**

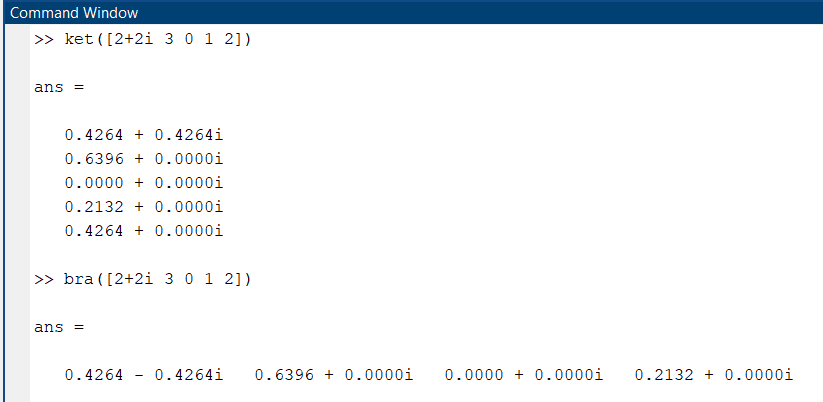
**KET-**





**BRA-**

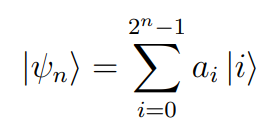




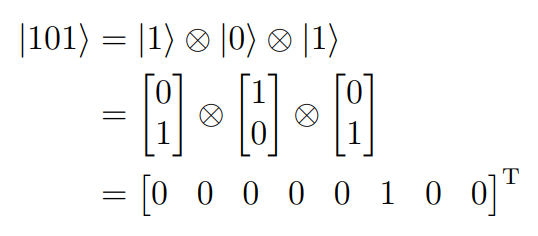
**Quantum Registers**

In quantum computing, a quantum register is a system comprising multiple qubits. A mathematical description of a quantum register is achieved by using tensor products of qubit bra or ket vectors.

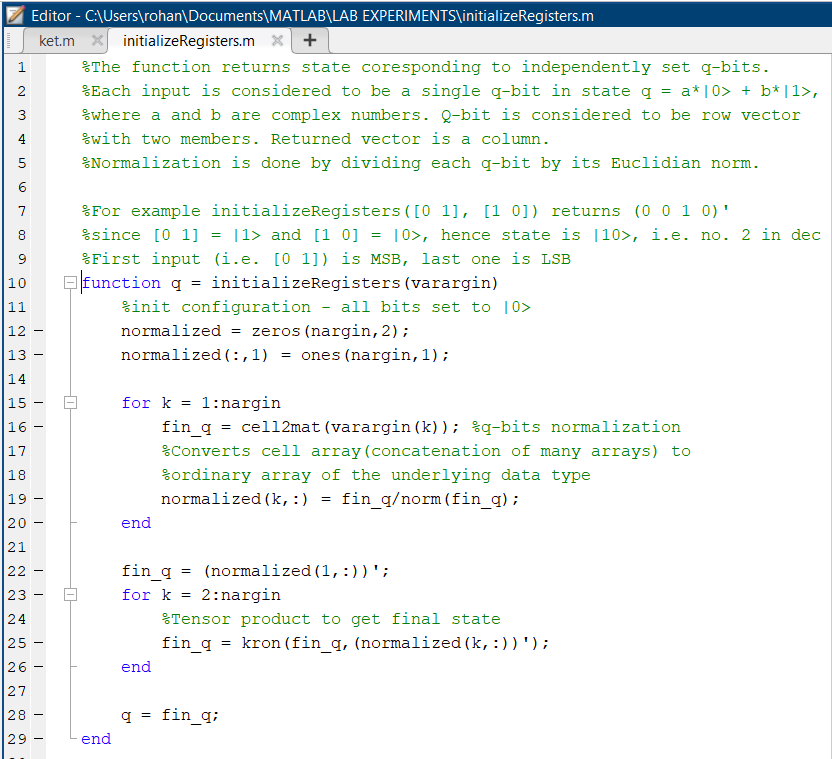
When collapsed, quantum registers are bit strings whose length determines the amount of information they can store. In superposition, each qubit in the register is in a superposition of |1> and |0>, and consequently a register of n qubits is in a superposition of all 2n possible bit strings that could be represented using n bits. The state space of a size-n quantum register is a linear combination of n basis vectors, each of length 2n:

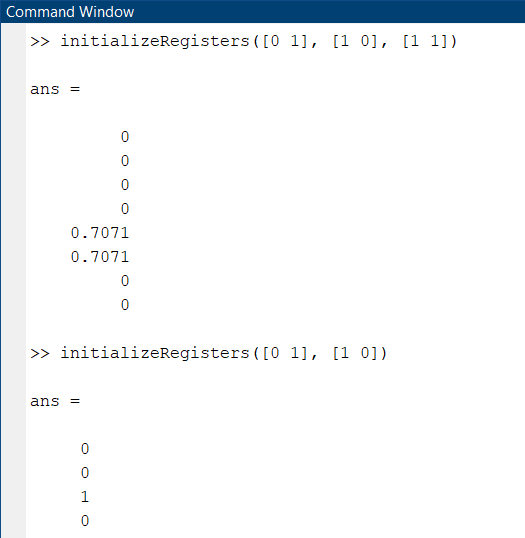


Here ‘*i’*  is the base-10 integer representation of a length-n number in base-2. Each possible bit configuration in the quantum superposition is denoted by the tensor product of its counterpart qubits. Consider |101>, the bit string that represents the integer value 5:



**Implementation in MATLAB**





**ENTANGLEMENT AND POWER OF QUANTUM COMPUTING**

Entanglement plays a crucial role in quantum computing. 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 cannot be described independently from one another. Altering the state of one particle will affect 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.

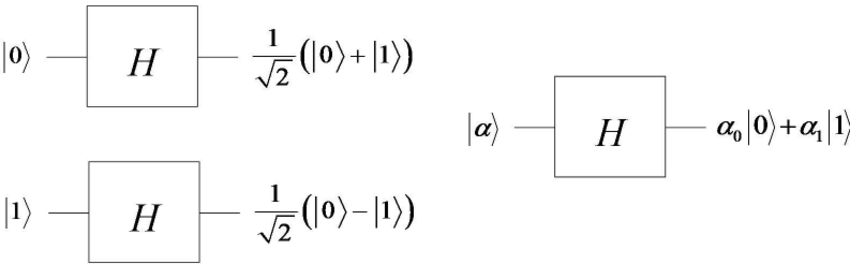
The complete utilization of Quantum properties gives us the following result-

* There’s a vast phase-space (a space in which all possible states of a system are represented)
* The phase space grows exponentially with the number of the entangled qubits, and all of the parameters representing that phase space are manipulated at once during a quantum computation step
* But reading out after the steps will ‘collapse’ the quantum state and we only see the projection of the state, i.e. a smaller and well defined set of solutions.

**QUANTUM GATES : HADAMARD**

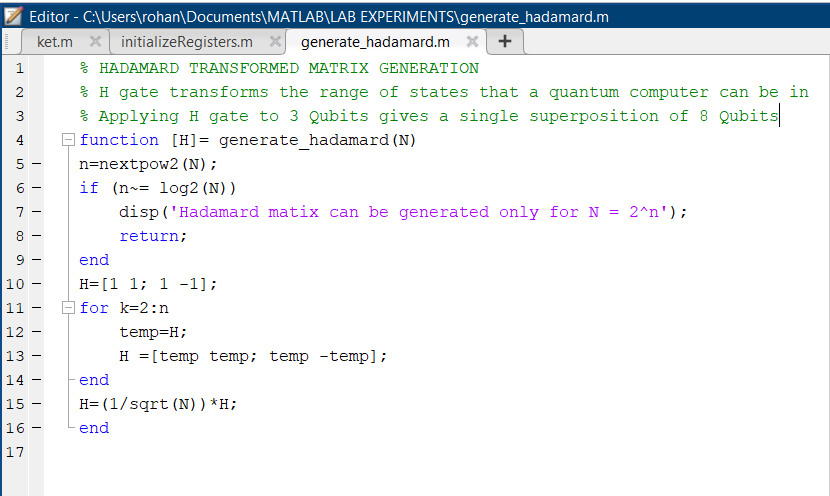
In classical computing, binary values, as stored in a register, pass through logic gates that, given a certain binary input, produce a certain binary output. Mathematically, classical logic gates are described using Boolean algebra. Quantum logic gates act in a similar way, in that quantum logic gates applied to quantum registers map the quantum superposition to another, together allowing the evolution of the system to some desired final state, a correct answer.

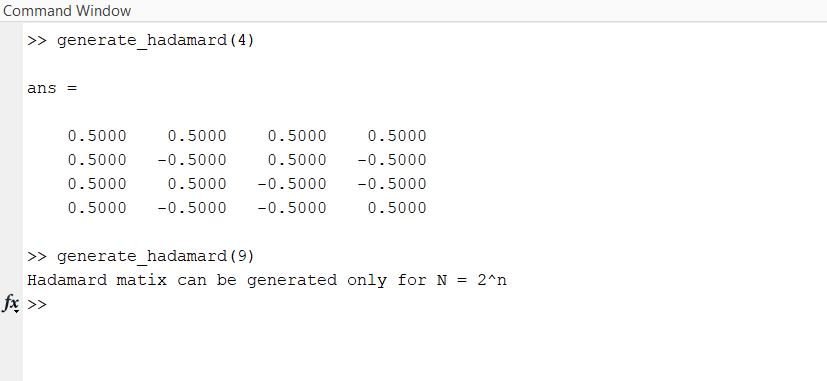
Quantum logic gates are mathematically represented as transformation matrices, or linear operators, applied to a quantum register by tensoring the transformation matrix with the matrix representation of the register. All linear operators that correspond to quantum logic gates must be unitary. That is, if a complex matrix U is unitary, then it must be true that U−1 = U†, where U† is the conjugate transpose: U† = UT.



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.

**Implementation in MATLAB**





**Introduction to SHOR’S ALGORITHM**

Shor’s algorithm solves the prime factorization problem (Given a integer N, find exactly two primes p and q such that N = pq) for large integers (N) in polynomial time.

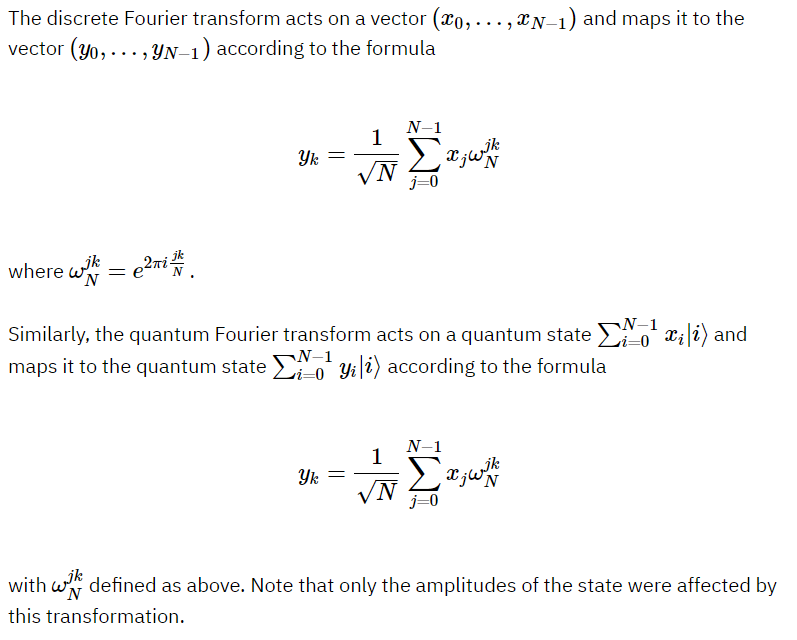
It does this, by finding the period, r, of some superposed periodic function f(x) = ax (mod N) and then applying classical algorithms to find a factor corresponding to the period. If r is even, then for some integer x, we can solve gcd(xr/2 -1, N) and gcd(xr/2 + 1, N) which will ideally give a factor of N. If r is not even then xr/2 ± 1 will not be an integer which means we do not have 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 important task that Quantum Computing accomplishes is finding the period r in smallest possible time.

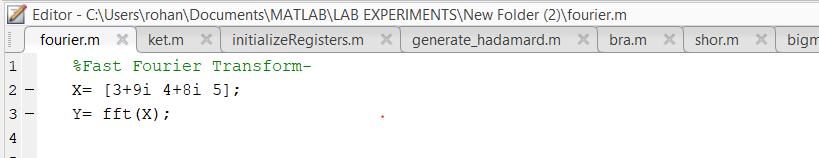
Shor’s Algorithm can be divided into two parts. The first part takes the prime factorization problem and turns it into the period finding problem which can be implemented on a classical computer. The second part of the algorithm finds the period using the Quantum Fourier Transform. It would seem that 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.

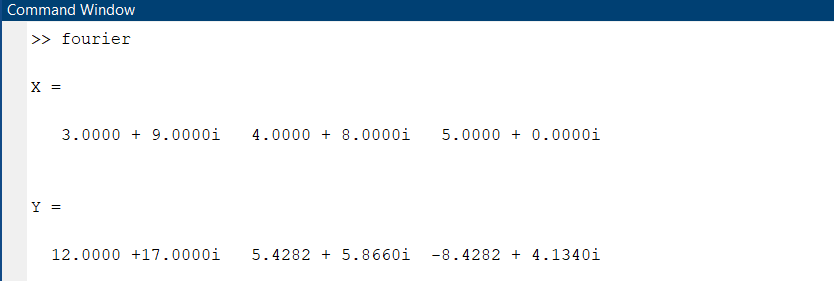
**QUANTUM FOURIER TRANSFORM**

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, and then outputs the transformed data.



Matlab has the inbuilt function fft( ) to perform Discrete Fourier Transform





**Shor’s Algorithm**

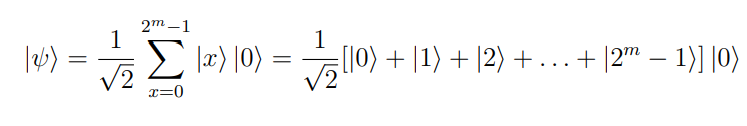
In brief, the Shor’s Algorithm as follows-

1. The integer to factorize is N.

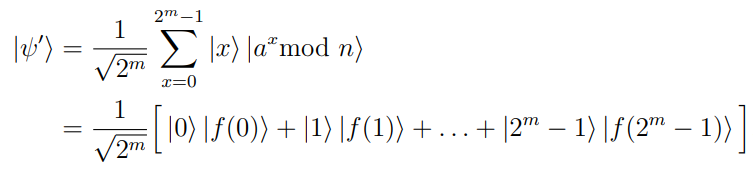
2. Choose a random integer a such that 1 < a < sqrt(N) and gcd(a, N) = 1.

3. Create two Quantum registers. Initialize register 1 and register 2 of size 2m both to state |0> so that the composite system has the state |ψ> = |0> |0>. For now, we will use this tensor product notation to distinguish between registers, e.g. if we initialize register 1 to |a> and register 2 to |b>, then the composite system has the state |ψ> = |a> |b>. We can think of a composite quantum system as the tensor product of multiple qubits e.g. |ψ1> ⊗ |ψ2> ⊗ |ψ3> ⊗ . . . ⊗ |ψm>.

4. 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 2m qubits normalized by 1/√2m . So applying the Hadamard gate to m qubits in register 1 leaves the computer in the superposed quantum state



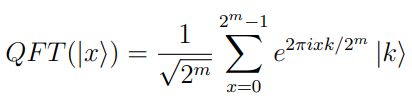
5. Compute f(x) = ax (mod n) for 0 ≤ x < 2m which will be:



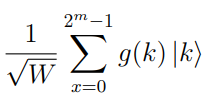
Leave the results in register 2.

6. Make a measurement on register 2. By doing so we will discover that register 2 will be in some base state |ax (mod n)>. By measuring register 2 we are collapsing the superposed quantum state into a smaller superposed quantum state.

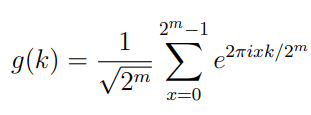
7. Apply the QFT to our newly measured quantum state |x, f (u)> as follows



8. From this we obtain the sum



where W is the total number of qubits in our measured quantum state which was made in step 6 of this procedure. The function g(k) is given by

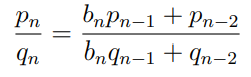


which is just the DFT of some binary sequence.

9. Choose one of the 2m 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 2m. After we make a measurement and apply the QFT, the quantum computer will only output a single point from the 2m 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 2m 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.

10. 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 P / 2m. Once we obtain the continued fraction, find the convergent values, [b0, b1,. . . bn] by calculating



Where p0 = b0, p1 = b1b0 + 1, q0 = 1 and q1 = b1. 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 ar ≡ 1 mod n.

11. Apply the exponent factorization method to factor n using the new found period r. Write the period as r = 2km where k ≥ 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 12until a factor of n is found.

OUTPUT OF OUR PROGRAM ON SHOR’S ALGORITHM IN MATLAB-



**Conclusion**

In this project the fundamentals of quantum phenomena were presented and applied in Quantum Computing. A library for Quantum Algorithms was created in MATLAB. This library can be used for various Quantum Algorithms like Grover’s Algorithm, Bernstein Vazirani, Deutsch-Josza and many others. I have implemented Shor’s Algorithm using this library. The mathematics behind Shor’s algorithm has also been explained thoroughly.

**Bibliography**

* Quantum Computation and Information (by Michael A. Nielsen and Isaac Chuang)
* Introduction to Quantum Mechanics by (D.J. Griffith)
* [www.robots.ox.ac.uk/~parg/pubs/qcf.pdf](http://www.robots.ox.ac.uk/~parg/pubs/qcf.pdf)

(Quantum Computing Functions Library)

* [www.quantiki.com](http://www.quantiki.com) (Shor’s Algorithm, Quantum Gates)