A Classical Mimic of Shor's Algorithm

Abhishek Roy[1]
[1]San Jose State University

December 13, 2024

1 State Representation

1.1 Information in a Qubit

Usually, in quantum mechanics, superpositions are represented as

$$|\Psi\rangle = a\,|0\rangle + b\,|1\rangle$$

where the basis vectors of the representation space (aka the Hilbert space) are $|0\rangle$ and $|1\rangle$, and their coefficients square to the probability of that ket state being measured [2].

To be able to do calculation with these states, and in particular linear algebra calculations, these states are represented as a linear combination of unit vectors, where the unit vectors are the basis vectors.

$$|\Psi\rangle = a\begin{bmatrix}1\\0\end{bmatrix} + b\begin{bmatrix}0\\1\end{bmatrix}$$

Let's take the example of an electron with spin up for $|0\rangle$ and spin down for $|1\rangle$. The superposed quantum state of the electron is a qubit. One thing to note would be that since a and b can be any complex number (as long as Ψ is normalized), it would almost seem that a qubit can represent a lot more than just 1 bit of information like a classical bit. That is not the correct way to look at it. Information must be considered as what can be measured from the system, whether quantum or classical. What can be measured is what can be used, which is why it is important classify information this way. When you measure the electron's spin at one time, you can only measure either spin up or spin down, not the probability of each measurement, nor the complex number representing the probability of each measurement—only spin up or spin down. Thus, we see that even a quantum bit can only give us one bit of information.

1.2 More Than 0 and 1

This now brings up the question, how to represent numbers larger than 0 and 1? Well, one way to think about it is that up until now, we have used a vector space with a dimension equal to the amount of distinct numbers we have had to represent—0 and 1. For bigger sets of distinct numbers to represent, we can use a bigger representation space; for 4 numbers $0, 1, 2, 3$, we can use a 4 dimensional vector space which may look like

$$|\Psi\rangle = \text{span}\left(\left\{\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}\right\}\right),$$

where the field is that of complex numbers. This can be extended into more dimensions.

The problem that now arises is how to make sense of this system physically, because there isn't any electron that has 4 different spin states that can represent 4 different numbers. The solution to this problem lies in the binary representation of decimal numbers. Like all number systems, the binary system is formed by permutations of 0s and 1s where each permutation represents a decimal number. Thus, borrowing from this idea of using multiple one-bit physical system permutations to represent decimal numbers, we can permute multiple one-bit electrons with up or down spin to represent decimal numbers. The way that permutations are represented in quantum mechanics is through the tensor product.

$$\begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} = |1\rangle \otimes |0\rangle = |10\rangle = |2\rangle_{\text{decimal}}$$

Thus, any decimal number can be represented through the ket states of the binary 1s and 0s. This also means that the dimension of the resulting representation vector space can only be 2^n where n is the number of qubits being used in the quantum system/computer. This fact will be important when creating the main matrix used in the code.

2 Shor's Algorithm

2.1 The Factoring Problem

Most of internet security, RSA encryption, is based on one thing: how hard it is to factor a number into its only two prime factors (assuming that the number has only two prime factors). While you can cycle through every number from 2 to the square root of the number, that can be very time consuming for the multiple-hundred-digit numbers used for internet security. Thus, the way a quantum computer can solve this problem comes from a different approach to solving the problem—modular arithmetic [3].

Given the number N, the number to be factored, take a guess integer (any integer) g. The two numbers will follow the following behavior

$$g^r = Nm + 1,$$

where r and m are just the integers that make this equation true. We can rearrange the equation to get

$$(g^{r/2} + 1)(g^{r/2} - 1) = Nm.$$

Ultimately, this equation will be used to figure out the prime factors of N since each of the terms $(g^{r/2}+1)$ and $(g^{r/2} - 1)$ will have common factors with N which will finally allow us to find those factors.

However, the part of the arithmetic that Shor's algorithm exploits is actually the periodicity of the power that g is raised to. In other words, let's say the following is true

$$g^r = Nm + x$$

where the remainder is x rather than 1. Then, there are other numbers for the power of g that satisfy the equation. Namely

$$g^{r+\alpha p} = Nm + x$$

where αp is just a whole number multiple of p which is the periodicity of this behavior. This implies that for our purposes with $x = 1$, we can basically set r to 0 and focus on finding the periodicity p. Although this isn't exactly what happens, the main point to be conveyed is that the periodicity is very important to making $g^{\square} \pm 1$ a good guess for a multiple of N.

The way that quantum mechanics plays into this is that periodicity is a wavelike property. Thus, a Fourier transform can be pretty effective in deciphering the problem as we will later see. Quantum mechanics has its own version of the Fourier transform called the Quantum Fourier Transform (QFT), which will help us find p; the QFT can be implemented in a quantum computer which makes quantum computers a very reliable tool in breaking encryption that is based on the difficulty of factoring the product of two primes.

2.2 Step 1: Modular Arithmetic

For the following analysis, all numbers are going to be represented in the decimal system of representation {0 through 9} unless otherwise specified. The coefficients of all the superposition states are going to be dropped as they are a hassle to type out. Just assume that each superposition state has an equal probability of being measured and that the wave-function is normalized. The example number we are using is $N = 15$, and our guess $g = 13$. We know 15 takes 4 bits to represent in binary so $n = 4$.

We start with $2n$ bits, all in the $|0\rangle$ state. We can maybe represent that as

$$|\Psi\rangle = |0\rangle_{\text{binary}}^{\otimes^n} |0\rangle_{\text{binary}}^{\otimes^n} = |0\rangle\,|0\rangle$$

with a separation to represent the two sets of n qubits. The first step we need to take is to apply a Hadamard gate to to each qubit in the first set of n qubits to get

$$|\Psi\rangle = \left(|0\rangle_{\text{binary}}^{\otimes^n} \otimes \mathbf{H}^n\right)|0\rangle_{\text{binary}}^{\otimes^n} = (|0\rangle + |1\rangle + \cdots + |2^n - 1\rangle)\,|0\rangle$$

$$= (|0\rangle + |1\rangle + |2\rangle + |3\rangle + |4\rangle + |5\rangle + |6\rangle + |7\rangle + |8\rangle + |9\rangle + |10\rangle + |11\rangle + |12\rangle + |13\rangle + |14\rangle + |15\rangle)\,|0\rangle$$

What this step does is it transforms the set of qubits from the $|0\rangle$ state to a superposition state; specifically, a superposition of all the numbers that can be represented by the n qubits $\left(|0\rangle + |1\rangle + \cdots + |2^4 - 1\rangle\right)$.

Now, we bring into action the second set of n qubits. Note, that currently, our state is

$$|\Psi\rangle = \left(|0\rangle\,|0\rangle + |1\rangle\,|0\rangle + \cdots + |2^4 - 1\rangle\,|0\rangle\right)$$

and is a **non-entangled** state. A non-entangled state is one which can be written as a tensor product, which it definitely can be.

The next step we are going to take is to create an **entangled state**, one that cannot be written as a tensor product. We are going to entangle the state in a very special way that involves g and N and how they are related. After entangling each of the second set of qubits to each of the first set of qubits, we get

$$|\Psi\rangle = \sum_{i=0}^{2^4-1} |i\rangle\,|g^i(\text{mod}N)\rangle = \left(|0\rangle\,|13^0(\text{mod}15)\rangle + |1\rangle\,|13^1(\text{mod}15)\rangle + \cdots + |2^4 - 1\rangle\,|13^{2^4-1}(\text{mod}15)\rangle\right)$$

$$\begin{aligned}
&= |0\rangle\,|1\rangle + |1\rangle\,|13\rangle + |2\rangle\,|14\rangle + |3\rangle\,|7\rangle \\
&+ |4\rangle\,|1\rangle + |5\rangle\,|13\rangle + |6\rangle\,|14\rangle + |7\rangle\,|7\rangle \\
&+ |8\rangle\,|1\rangle + |9\rangle\,|13\rangle + |10\rangle\,|14\rangle + |11\rangle\,|7\rangle \\
&+ |12\rangle\,|1\rangle + |13\rangle\,|13\rangle + |14\rangle\,|14\rangle + |15\rangle\,|7\rangle\,.
\end{aligned}$$

Since this state cannot be written as a tensor product, this is an entangled state.

Up to now, all the calculations we have done are purely classical and could probably be done faster on a classical computer. The next step is where quantum mechanics begins to show up and help us. What we do is measure the second set of qubits which currently represent the second kets from the equation above. In quantum mechanics, no matter what interpretation one chooses, one thing is for certain, the wave function seems to collapse into one measurement. In this case, if the second set of n qubits were measured to be $|0111\rangle_{\text{binary}}$, we'll have measured the second ket to be $|7\rangle$. Thus, our wave-function will have collapsed to the non-entangled state of

$$|\Psi\rangle = |3\rangle\,|7\rangle + |7\rangle\,|7\rangle + |11\rangle\,|7\rangle + |15\rangle\,|7\rangle = (|3\rangle + |7\rangle + |11\rangle + |15\rangle)\,|7\rangle\,.$$

As you can probably notice, the numbers represented in $|\Psi\rangle$ do wave a periodicity to them which is what we want to extract. However, if we were to measure now, we would get either 3, 7, 11, or 15 which wouldn't help us a lot. Even if we tried doing the process again to get another number to compare, there is no guarantee we'll have measured the second set of qubits to be 7 again.

Also, for further calculations, I will drop the $|7\rangle$ in the end since it doesn't do much for our calculations at all.

2.3 Step 2: The Quantum Fourier Transform

The Quantum Fourier Transform is what extracts the period from our current wave-function

$$|\Psi\rangle = (|3\rangle + |7\rangle + |11\rangle + |15\rangle)\,|7\rangle\,.$$

The QFT is defined as such

$$\text{QFT}\,|x\rangle = \sum_{y=0}^{2^n-1} \left(\exp\frac{2\pi i x y}{2^n}\right)|y\rangle\,.$$

One way of seeing what the QFT does is to say that it encodes a period x into the wave-function by putting in a multiple of x for the phase of each superposition state's coefficient. Furthermore, those

multiples of x that are put in for a certain superposition state are directly related to that superposition state since the value of that state is what is multiplying x.

One thing to note, however, is that the above equation takes a period and encodes it into a wave-function. What we want to do is the inverse. We want to take a wave-function state, and output the period. To do this, we simply take the inverse of the QFT by adding a negative sign to the exponent. We get

$$\mathrm{QFT}^{-1}\left|\tilde{x}\right\rangle = \sum_{y=0}^{2^n-1}\left(\exp\frac{-2\pi i\tilde{x}y}{2^n}\right)\left|y\right\rangle,$$

where $\tilde{x}$ is just a notation used for the inverse function's input; it is still a number just like x.

Now, all we have to do is take the QFT inverse of the wave-function we have, to get a wave-function of periods.

$$\mathrm{QFT}^{-1}\left|\Psi\right\rangle = \sum_{y=0}^{2^4-1}\left(\exp\frac{-2\pi i3y}{2^n}+\exp\frac{-2\pi i7y}{2^n}+\exp\frac{-2\pi i11y}{2^n}+\exp\frac{-2\pi i15y}{2^n}\right)\left|y\right\rangle.$$

The end result of the calculation looks like

$$\left|\Psi\right\rangle = 4\left|0\right\rangle + 4i\left|4\right\rangle - 4\left|8\right\rangle - 4i\left|12\right\rangle,$$

where every possible outcome of measurement on $|\Psi\rangle$ is a multiple of the periodicity of r in the equation $g^r = Nm + x$.

2.4 Step 3: Finishing up the Arithmetic

The last part of solving the factoring problem is rather simple. After making the measurement on the current $|\Psi\rangle$ state, we use the equation

$$j\frac{2^n}{r} = \text{measurement}$$

where j is the first integer that satisfies the equation. Remember, r is the power of g that we need to plug into $(g^{r/2}+1)(g^{r/2}-1) = Nm$, and p is represented in the measurement itself because the measurement is a multiple of the periodicity. Solving for r we get

$$r = j\frac{2^n}{\text{measurement}} = j\frac{2^4}{4}$$

supposing that we measure $|4\rangle$ from the wave-function. Since $j = 1$ gives an integer value for r, we find that $r = 4$.

Thus

$$r = 4$$
$$g^{r/2}+1 = 13^{4/2}+1 = 170$$
$$g^{r/2}-1 = 13^{4/2}-1 = 168$$

and finding the greatest common factors of each of those numbers with N the product of the two primes gives us

$$\gcd(170, 15) = 5$$
$$\gcd(168, 15) = 3$$

which gives us exactly the numbers we are looking for.

3 Matrix Representation of the Quantum Fourier Transform

Taking from the idea of how a quantum state can be represented as a column vector, we know that with n qubits, the size of the column vector would be $2^n \times 1$. Thus, the matrix that represents the inverse Quantum Fourier Transform must be $2^n \times 2^n$ so it can multiply a column vector state and output a column vector of the same size representing its transformed state. Furthermore, it can easily be seen from the equation for the QFT^{-1} that it is a linear transformation, thus it *has* to have a matrix representation.

Let's construct the matrix representation of the QFT^{-1} using what we know about linear transformations. For the linear transformation $\text{QFT}^{-1} : \mathbb{C}^{2^n} \to \mathbb{C}^{2^n}$, we know the linear transformation matrix has to be of the form

$$\text{QFT}^{-1} = \left[\text{QFT}^{-1}(|0\rangle), \text{QFT}^{-1}(|1\rangle), \cdots, \text{QFT}^{-1}(|2^n - 1\rangle)\right]$$

where $|0\rangle, |1\rangle, \cdots, |2^n - 1\rangle$ are the basis vectors of the state space. Since we know the column vector representation of the basis ket vectors and thus also the $\text{QFT}^{-1}|\tilde{x}\rangle$ (because the transform is defined as a linear combination of the same basis ket vectors), we get the following

$$\text{QFT}^{-1} = \left[\text{QFT}^{-1}\left(\begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix}\right), \text{QFT}^{-1}\left(\begin{bmatrix}0\\1\\0\\\vdots\\0\end{bmatrix}\right), \cdots, \text{QFT}^{-1}\left(\begin{bmatrix}0\\0\\0\\\vdots\\1\end{bmatrix}\right)\right]$$

$$\text{QFT}^{-1}|\tilde{x}\rangle = \begin{bmatrix}\exp\frac{-2\pi i\tilde{x}\cdot 0}{2^n}\\ \exp\frac{-2\pi i\tilde{x}\cdot 1}{2^n}\\ \vdots \\ \exp\frac{-2\pi i\tilde{x}\cdot(2^n-1)}{2^n}\end{bmatrix}$$

$$\text{QFT}^{-1} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \exp\frac{-2\pi i\cdot 1\cdot 1}{2^n} & \exp\frac{-2\pi i\cdot 2\cdot 1}{2^n} & \cdots & \exp\frac{-2\pi i\cdot(2^n-1)\cdot 1}{2^n} \\ 1 & \exp\frac{-2\pi i\cdot 1\cdot 2}{2^n} & \exp\frac{-2\pi i\cdot 2\cdot 2}{2^n} & \cdots & \exp\frac{-2\pi i\cdot(2^n-1)\cdot 2}{2^n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \exp\frac{-2\pi i\cdot 1\cdot(2^n-1)}{2^n} & \exp\frac{-2\pi i\cdot 2\cdot(2^n-1)}{2^n} & \cdots & \exp\frac{-2\pi i\cdot(2^n-1)\cdot(2^n-1)}{2^n} \end{bmatrix}$$

Since the QFT^{-1} matrix is a linear transformation matrix, and $|\Psi\rangle$ is just a linear combination of the basis vectors that QFT^{-1} is transforming, we can directly apply the transformation matrix on the column representation of $|\Psi\rangle$ by multiplying the matrix on the left.

4 Code: Test Results and Efficiency

4.1 Making of the Code

Much of the code for the classical mimic of Shor's algorithm was easy to implement and non-quantum. For example in step 1, we don't even need to apply a Hadamard gate operator on each of the first n qubits because the final result is just the superposition of all the possible basis quantum states. In other words, instead of applying n Hadamard gates or the tensor product of n Hadamard gates, we can just make a column vector that has all entries as 1.

Furthermore, the second set of n qubits can just be a different column vector, the entries of which we keep track of. The entangling process that the quantum computer goes through is not needed at all for our classical counterpart. All we need to do is keep track of the modulus values in a second array also of size $2^n \times 1$ and keep track of the unique values in a separate python set. Then, we can select a random value from the set, and zero out all indices from the original array (the Hadamard product) that aren't the indices matching with the selected random value in our array of modulus values. The zeroed out array is exactly the column vector to be inputted into the inverse QFT. **This is where the inverse QFT matrix is multiplied by the column vector.** This is exactly what the first two thirds of `quantum_part.py` does. (All column vectors, matrices are made and multiplied by `numpy` [1] software.)

To get the inverse QFT matrix itself is quite easy. Looking at `quantum_fourier_transform.py`, we see that all I did was create separate column vectors that are the columns of the QFT^{-1} matrix, add them to a tuple, and have `numpy` [1] splice the columns together into a full matrix.

For the final step of actually measuring a multiple of the periodicity of the behavior of the main equation, the transformed quantum vector dictates a biased probability to select some state (whose square modulus isn't zero) from the vector. I use `scipy` [4] to find the norm of the column vector, and use that norm to create a biased probability distribution with I then feed into python's `random_value` function.

4.2 Tests

Because this version of the algorithm requires a multiplication of a $2^n \times 2^n$ transformation matrix with a $2^n \times 1$ column vector, not to mention the time it takes to generate the transformation matrix itself, it is quite slow compared to an algorithm that just brute forces the calculations and compares almost every number between 1 and N.

To factor small numbers like 21, the algorithm took 0.09 seconds. As N, the number to factor, got bigger with factors like 19 and 17, I had to start using the `decimal` library in python to deal with the bigger numbers. However, with 19 and 17, the algorithm was surprisingly fast, taking only 0.2 seconds.

When I started to test with bigger numbers like 97 and 89, that is when things got very slow. The algorithm took a total of 21 minutes to factor 8663 (97×89). This is when I decided to use a faster algorithm.

4.3 Improvements

One thing that I realized for main part of the algorithm, multiplying the transformation matrix by the column vector, was that every entry in the column vector is just 0 or 1. So I could save a lot of time by not unnecessarily making columns which will not contribute at all in the end. So, when creating the faster version of the algorithm, I just scrapped the use of the `quantum_fourier_matrix` method which created the transformation matrix. Instead, I took the column vector of just 1s and 0s, and made the transformation columns for the index elements with a value of 1. Then, I just added those columns together. This faster version of the algorithm is surprisingly faster than the older version. It factored the product of 97 and 89 in just 21 seconds, compared to the previous 9 minute time.

The main reason for the column vector only being made of 1s and 0s is the fact that in our representation system, the basis vectors include every number that can be represented by the qubits. There is no number that needs to be represented by an addition of vectors from another number system, in fact, addition is only reserved to represent quantum superposition states. Thus, since the basis includes every number we could possible want to represent, the column vector to multiply is just a linear combination of standard unit vectors which is very easy to work with computationally. Another thing that helped would be the fact that I didn't include the magnitude of the probability amplitudes until the very end. Until then, all I needed to work with would be 1s and 0s instead of fractions and irrational numbers, which is also a computationally efficient strategy.

References

[1] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[2] Leonard Susskind and Art Friedman. *Quantum mechanics: the theoretical minimum.* Basic Books, 2014.

[3] CH Ugwuishiwu, UE Orji, CI Ugwu, and CN Asogwa. An overview of quantum cryptography and shor's algorithm. *Int. J. Adv. Trends Comput. Sci. Eng*, 9(5), 2020.

[4] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris,

Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.