Math and proofs for the quantum algorithms implemented in the code

Contents

1 Important notes

A few things that cause weird behavior;

- if you get probabilities that sum to more than one, you probably applied a controlled gate with repeated qubits; i.e. apply a controlled-not gate with control and target both being the same qubit.
- Be very careful with operations involving unsigned ints and ints. I decided to make most types in the code unsigned, but this often causes strange things to happen, and caused me many minutes of confusion. If in doubt, be very explicit; convert everything to an int when doing loops and calculations, and then convert back to unsigned int when sending values into functions.
- When computing modulo powers, use my `mod_power(a, x, N)` function from methods.h instead of `(int)pow(a, x) % N`, because the latter often causes an overflow if `a` or `x` are large enough. The `mod_power` function works based off of Theorem 3.

2 Grover search algorithm

Let $\{|x\rangle \mid 0 \leq x < N\}$ be a basis in order (for the code, $|0\rangle$ corresponds to $|00..00\rangle$, $|2\rangle$ corresponds to $|00..10\rangle$, $|N-1\rangle$ corresponds to $|11..11\rangle$, ...)

Consider that we are given some unitary operator U_ω such that

$$U_\omega |x\rangle = \begin{cases} -|x\rangle & \text{if } x = \omega \\ |x\rangle & \text{if } x \neq \omega \end{cases} = I - 2|\omega\rangle\langle\omega|$$

Our goal is to find ω (U_ω is a 'black box', we cannot look inside). Let $|s\rangle$ be the equal superposition of all basis states $|s\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$, and the Grover diffusion operator be defined as $D = 2|s\rangle\langle s| - I$.

2.1 Steps

1. Start with the register in state $|\psi\rangle = |0\rangle = |00...00\rangle$; there must be at least $\log_2(N)$ number of qubits in order to have the necessary N basis states
2. Apply the Hadamard gate to each qubit so that $|\psi\rangle = |s\rangle$
3. Apply U_ω to the system $|\psi\rangle$, then apply D to the system $|\psi\rangle$
4. Perform step 3 a total of r times, where r is the closest integer to $\frac{\pi}{4\arcsin\left(1/\sqrt{N}\right)} - 1/2$
5. Perform a measurement to collapse the system into a basis state; the resulting basis state will, with high probability, be $|\psi\rangle = |\omega\rangle$

2.2 Why it works

Let $|s'\rangle = \frac{1}{\sqrt{N-1}} \sum_{x=0,\neq\omega}^{N-1} |x\rangle$ so that $|s\rangle = \sqrt{\frac{N-1}{N}}|s'\rangle + \frac{1}{\sqrt{N}}|\omega\rangle$. $|s'\rangle$ and $|\omega\rangle$ are orthonormal; use them to define an orthonormal basis $\{|s'\rangle, |\omega\rangle\}$. In this basis,

$$\langle s| = \begin{pmatrix} \sqrt{\frac{N-1}{N}} & \frac{1}{\sqrt{N}} \end{pmatrix} \qquad \Rightarrow D = \frac{1}{N}\begin{pmatrix} N-1 & 2\sqrt{N-1} \\ 2\sqrt{N-1} & -(N-2) \end{pmatrix} \qquad U_\omega = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Thus, the total operator that we define in step 3 above is

$$DU_\omega = \frac{1}{N}\begin{pmatrix} N-2 & -2\sqrt{N-1} \\ 2\sqrt{N-1} & N-2 \end{pmatrix}$$

Notice that this can be written as the standard rotation operator

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

with

$$\begin{aligned} \frac{N-2}{N} &= \cos\theta \\ &= \cos^2(\theta/2) - \sin^2(\theta/2) \\ &= 2\cos^2(\theta/2) - 1 \quad \longrightarrow \end{aligned} \qquad \begin{aligned} \frac{2\sqrt{N-1}}{N} &= \sin\theta \\ &= 2\sin(\theta/2)\cos(\theta/2) \\ &= 2\sin(\theta/2)\sqrt{\frac{N-1}{N}} \end{aligned}$$

where we plug in the result of $\cos(\theta/2)$ on the third line on the left into the third line on the right. Thus, at every iteration, we rotate our state $|\psi\rangle$ by an angle $\theta = 2\arcsin\left(1/\sqrt{N}\right)$ counterclockwise in the $(|s'\rangle, |\omega\rangle)$ plane.

Note that we start off rotated counterclockwise at an angle ϕ in the $(|s'\rangle, |\omega\rangle)$ plane, where $\cos\phi = \langle s|s'\rangle = \sqrt{\frac{N-1}{N}}$. This is equivalent to $\sin\phi = 1/\sqrt{N}$, or $\phi = \theta/2$. We apply the operation DU_ω r times to our state $|\psi\rangle$ that is initially at $|s\rangle$. The goal is to make it that $\langle\omega|\psi\rangle = \langle\omega|\,(DU_\omega)^r\,|s\rangle$ is maximum. Thus, we need to rotate an angle of $\pi/2 - \phi$, because we start off offset from $|s'\rangle$ by ϕ and we want to get to $|\omega\rangle$ which is offset from $|s'\rangle$ by $\pi/2$.

Therefore, we want that $r\theta = \pi/2 - \phi$. Plugging in for $\phi = \theta/2$ and $\sin(\theta/2) = 1/\sqrt{N}$, we find that

$$r = \frac{\pi}{4\arcsin\left(1/\sqrt{N}\right)} - \frac{1}{2}$$

Of course, we need to apply it an integer number of times, so we pick the closest integer to r. Note that for large N, $r \approx \pi\sqrt{N}/4$.

Applying the Hadamard gate to each qubit at the beginning is how we initialize the initial $|\psi\rangle = |s\rangle$ state.

3 Quantum Fourier transform (QFT)

The quantum Fourier transform is very similar to the classical DFT. It is defined to be the unitary operator F that operates on the basis states as follows;

$$F\,|j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \exp(2\pi i jk/N)\,|k\rangle$$

With n qubits, there are $N = 2^n$ basis states. We can see then that in this basis, the matrix of F is defined component wise by

$$F_{jk} = \frac{\omega^{jk}}{\sqrt{N}} \qquad \text{where } \omega = \exp(2\pi i/N)$$

I will not go into detail on how this transformation can be mapped to a series of one and two qubit logic gates; [1] pages 217-220 has a very good explanation, and I fear I would just plagiarize if I tried to explain it here.

In the end, we apply the Hadamard gate $\mathcal{O}(n)$ times and the Controlled-Phase gate $\mathcal{O}(n^2)$ times, then the swap gate $\mathcal{O}(n/2)$ times (see below). The algorithm can be seen in the code.

3.1 Swap gate

When swapping two qubits, we have the basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$, so the operations corresponding to the first and last basis element should be the identity, and the operations corresponding to the middle two should be the NOT gate. Thus,

$$U_{\text{swap}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Consider the Controlled-Not gate in this basis. If we want to control with the first qubit and NOT the second, then the operations corresponding to the first two basis elements should be the identity, because the control is zero, and the operations corresponding to the second two should be the NOT gate. Call this U^{12}_{CNot}. If we want to control with the second qubit and NOT the first, then the operations corresponding to the first and third basis elements should be the identity because the control is zero, and the operations corresponding to the second and fourth basis elements should be the NOT gate. Call this U^{21}_{CNot}. Thus,

$$U^{12}_{\text{CNot}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad U^{21}_{\text{CNot}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

By matrix multiplication, we see that

$$U_{\text{swap}} = U^{12}_{\text{CNot}} U^{21}_{\text{CNot}} U^{12}_{\text{CNot}}$$

In other words, applying the swap gate to qubits i and j is equivalent to applying the Controlled-Not gate three times; the first and third time with the control being qubit i and target qubit j, and the second time with the control being qubit j and the target qubit i.

4 Shor factorization algorithm

Using quantum computers to compute the factors of a number.

4.1 Necessary lemmas and theorems

Theorem 1. *Define the order, or period, of* $a \mod N$ *to be the smallest* $r \geq 1$ *such that*

$$a^r \equiv 1 \mod N$$

ie $a^{x+r} \equiv a^x \mod N$. *If* a *and* N *are coprime, then a period* r *must exist and* $r < N$.

Proof. Consider the list $[a^1 \mod N, a^2 \mod N, ..., a^N \mod N]$. Since $\gcd(a, N) = 1$, 0 is not in this list, meaning that this list can only contain the numbers $(1, 2, ..., N-1)$. Thus we have a list of N elements but only $N-1$ possible numbers; by the Pigeonhole principle, the list must contain at least one repeated element, implying that there exists an $0 < m \leq N$ and an $0 < n < N$ with $m > n$ such that

$$a^n \mod N = a^m \mod N$$

Therefore, the period is $r = m - n$ where $0 < m - n < N$. □

Lemma 1. $\gcd(a, b) = \gcd(a, b + a)$

Proof. Let $g_1 = \gcd(a, b)$ and $g_2 = \gcd(a, b + a)$. Since $g_1|a$ and $g_1|b$, it must be that $g_1|a + b$. Similarly, since $g_2|a$ and $g_2|a + b$, it must be that $g_2|b$. Thus, $g_1 = g_2$. □

Theorem 2. *In order to compute* $\gcd(a,b)$*, we execute the following three assignments in order until* $b=0$*:*

$$t = b;\ b = rem(a/b);\ a = t;$$

When $b=0$*,* a *has become the gcd.*

Proof. Show that $g = \gcd(a,b) = \gcd(a, \text{rem}(b/a))$. From this, the algorithm is clear, as is it just a recursive occurrence.

We have that $a \equiv r \mod b$ where r is $\text{rem}(b/a)$. Thus, $a - r = zb$ for some integer z. By Lemma 1,

$$\gcd(b,r) = \gcd(b, r+b) = \gcd(b, r+2b) = ... = \gcd(b, r+zb) = \gcd(b,a)$$

□

Lemma 2. *Bezout's identity states that for any nonzero integers* a *and* b*, there exists integers* x *and* y *such that* $ax + by = \gcd(a,b)$*.*

Proof. This is easy to see; since $a/\gcd(a,b)$ and $b/\gcd(a,b)$ are both integers, we have that $xz_1 + yz_2 = 1$ where z_1, z_2 are integers. Thus, pick y to be an integer with $yz_2 \equiv 1 \mod z_1$, then $x = (1 - yz_2)/z_1$ is an integer. Note that we can pick such a y so long as $z_1 \neq z_2$. If $z_1 = z_2$, then $a = b$, and the identity is trivial. □

Theorem 3. *For positive integers* a*,* x*, and* N*,*

$$a^x \mod N = (a \mod N)^x \mod N$$

Proof. a can be written $a = a_0 + \lambda N$ where $a_0 < N$. Then,

$$\begin{aligned} a^x \mod N &= (a_0 + \lambda N)^x \mod N \\ &= (a_0^x + N(\text{stuff})) \mod N \\ &= a_0^x \mod N \\ &= (a \mod N)^x \mod N \end{aligned}$$

□

Lemma 3. *If* $N | b^2 - 1$*, then*

1. $\gcd(b-1, N) = 1 \longrightarrow b \equiv -1 \mod N$
2. $\gcd(b+1, N) = 1 \longrightarrow b \equiv 1 \mod N$

Proof.

1. By Lemma 2, there exists x and y such that $(b-1)x + Ny = 1$. Thus, $(b^2-1)x + N(b+1)y = b+1$. Since $N|b^2-1$, it must be that $N|b+1$ in order for that relation to hold. Thus, $b \equiv -1 \mod N$.
2. By Lemma 2, there exists x and y such that $(b+1)x + Ny = 1$. Thus, $(b^2-1)x + N(b-1)y = b-1$. Since $N|b^2-1$, it must be that $N|b-1$ in order for that relation to hold. Thus, $b \equiv 1 \mod N$.

□

Theorem 4. *Let* N *not be even or an integer power of a prime (modular arithmetic is often different when dealing with primes). If* a *is coprime with* N *(ie* $\gcd(a,N) = 1$*) and the period of* $f(x) = a^x \mod N$ *is* $r \in evens$ *with* $a^{r/2} \not\equiv -1 \mod N$*, then* $\gcd(a^{r/2}+1, N)$ *and* $\gcd(a^{r/2}-1, N)$ *are both nontrivial factors of* N*.*

Proof. Let r be the smallest integer such that $a^r \equiv 1 \mod N \quad \Rightarrow N|(a^r - 1)$.

Assume that we have found r and it is even. Define $b \equiv a^{r/2} \mod N$. We know that $a^{r/2} \not\equiv 1 \mod N$ because we found that r was the smallest integer such that $a^r \equiv 1 \mod N$. Thus, $b \not\equiv 1 \mod N$, and by assumption $b \not\equiv -1 \mod N$. (Note that there exists an a such that $a^{r/2} \not\equiv 1, -1 \mod N$ via the Chinese remainder theorem since N is not a prime power).

Thus, $d = \gcd(b-1, N)$ is a proper factor of N, because $1 < d < N$;

- if $d = 1$, then by Lemma 3, since $N|(a^r - 1 = b^2 - 1)$, $b \equiv -1 \mod N$, which contradicts our assumption.
- if $d = N$, then $N|b - 1$ meaning that $b \equiv 1 \mod N$ which contradicts what we found earlier.

Similarly, $f = \gcd(b + 1, N)$ is a proper factor of N, because $1 < f < N$;

- if $f = 1$, then by Lemma 3, since $N|(a^r - 1 = b^2 - 1)$, $b \equiv 1 \mod N$, which contradicts what we found earlier.
- if $f = N$, then $N|b + 1$ meaning that $b \equiv -1 \mod N$ which contradicts our assumption.

□

4.1.1 Continuous fractions expansion

The continuous fraction expansion of c is defined by

$$c = [a_0, a_1, a_2, a_3, ...] = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + ...}}}$$

Consider truncating this expansion:

$$\begin{aligned}
[a_0] &= a_0 = \frac{a_0}{1} \\
[a_0, a_1] &= a_0 + \frac{1}{a_1} = \frac{a_0 a_1 + 1}{a_1} \\
[a_0, a_1, a_2] &= a_0 + \frac{1}{a_1 + \frac{1}{a_2}} = \frac{a_0 a_1 a_2 + a_0 + a_2}{a_1 a_2 + 1} \\
... &= ...
\end{aligned}$$

Let n_i be the numerator and d_i be the denominator for

$$[a_0, a_1, ..., a_i] = \frac{n_i}{d_i}$$

From the above expansion, it is easy to see that (easier if you do one or two more expansions)

$$d_0 = 1 \quad d_1 = a_1 \qquad d_i = a_i d_{i-1} + d_{i-2}$$

$$n_0 = a_0 \quad n_1 = a_0 a_1 + 1 \qquad n_i = a_i n_{i-1} + n_{i-2}$$

Similarly, if $c = [a_0, a_1, ...]$ then

$$a_0 = \lfloor c \rfloor \quad \mathcal{E}_0 = c - a_0 \qquad a_{i+1} = \left\lfloor \frac{1}{\mathcal{E}_i} \right\rfloor \quad \mathcal{E}_{i+1} = \frac{1}{\mathcal{E}_i} - a_{i+1}$$

where $\mathcal{E}$ is defined to aid in calculating the recurrence.

4.2 Quantum period-finding method

We want to find the period of $f(x)$; ie r such that $f(x) = f(x + r)$. Assume that $0 \leq f(x) < N$ for all x (this is the only kind of function of interest for Shor's algorithm, which uses a function modulo N).

We will use two registers; the first with L_1 qubits, and the second with L_2 qubits.

We need to choose L_1 such that register one has enough bits to hold numbers from 0 to $\geq N^2$; pick $N^2 \leq 2^{L_1} \leq 2N^2$, so $2 \log_2 N \leq L_1 \leq \log_2(2N^2)$. Let $q = 2^{L_1}$. We want the first register to be able to hold $q \geq N^2$ numbers so that

$Q/r > N$, meaning that there are at least N different x that can be stored that produce the same $f(x)$ (see below for more detail)[1].

We need to choose L_2 such that register two has enough bits to hold numbers from 0 to $N-1$; so $L_2 = \lfloor \log_2 N \rfloor + 1$.

1. Register one has L_1 qubits; register two has L_2 qubits. Initialize $|\psi\rangle = |0\rangle\,|0\rangle$. Or, if you like, our register has $L_1 + L_2$ qubits, and we initialize
$$|\psi\rangle = |0, 0\rangle$$
I will use the ladder notation; there are L_1 qubits before the comma, and L_2 qubits after the comma.

2. Make a uniform superposition in the first register. This can either be done by apply the Hadamard gate to qubits 0 through $L_1 - 1$, or by applying the quantum Fourier transform to qubits 0 through $L_1 - 1$. Either way, we now have
$$|\psi\rangle = \frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |x, 0\rangle$$
As usual, $|x\rangle$ is our base ten number representation of the qubit representation of x in binary; ie $|010110...\rangle$.

3. Apply the oracle function to register two with inputs from register one: ie $|j, 0\rangle \to |j, f(j)\rangle$. Thus, we now have
$$|\psi\rangle = \frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |x, f(x)\rangle$$
This entangles the two registers! This oracle transformation all happens at once due to quantum parallelism, the reason quantum computers are of interest. In real life, I think that the quantum circuit implementation of the oracle must be created specifically for a given function f. Lucky for us, we're using C++, so we just use the apply_function method to the register.

4. Measure the second register. Say our measurement results in the second register being w (ie we measure qubits L_1 through $L_1 + L_2 - 1$ and convert to base ten and we get w). Then, because the registers are entangled, our state becomes
$$|\psi\rangle = \frac{1}{\sqrt{||\mathcal{X}||}} \sum_{x \in \mathcal{X}} |x, w\rangle \qquad \mathcal{X} = \{\text{all } x \text{ s.t. } f(x) = w\}$$

5. Now we apply the quantum fourier transform to the first register. Recall the QFT from Section 3; we have $\omega = \exp(2\pi i/q)$, then applying the QFT to ψ gives
$$|\psi\rangle = \frac{1}{\sqrt{||\mathcal{X}||}} \sum_{x \in \mathcal{X}} \frac{1}{\sqrt{q}} \sum_{k=0}^{q-1} \omega^{kx} |k, w\rangle$$

6. Measure register one. Now the whole system is collapsed; we are now in the state
$$|\psi\rangle = |k, w\rangle$$
The probability of measuring a state $|k, w\rangle$ (ie measuring the value k since we already collapsed register two – w is guaranteed) is
$$p_k = \frac{1}{q||\mathcal{X}||} \left| \sum_{x \in \mathcal{X}} \omega^{kx} \right|^2 = \frac{1}{q||\mathcal{X}||} \left| \sum_{x \in \mathcal{X}} \exp(2\pi i x k/q) \right|^2$$

[1] q is the number of numbers register 1 can hold. q must be at least $2N$ so that even if the period is $r = N - 1$ (which is the maximum period it can have according to Theorem 1), the first register can still hold enough numbers to have at least two x such that $f(x_0) = f(x_1)$ because $x_0 + r = x_1$. Most literature I have found says that we should initialize q such that $N^2 \le q \le 2N^2$ so that $q/r > N$ even when $r = N - 1$; thus there will be at least N different x such that $f(x_0) = f(x_1) = ... = f(x_N)$ where $x_i = x_0 + ir$. But for the code, I've found that $q = 2N$ works fine. By using a smaller q, we have a smaller probability of measuring r, but since simulating a quantum register on a classical computer is exponential, it is faster for the code to have a lower probability of measuring r and maybe having to try again.

We know that for each $x \in \mathcal{X}$, $x = x_0 + jr$ where x_0 and j are some integers and r is the period, because our state is collapsed such that all x satisfy $f(x) = w$. Thus, $f(x_0) = f(x_0 + jr)$. Therefore, the probability becomes

$$p_k = \frac{1}{q||\mathcal{X}||} \left| \sum_{j=1}^{||\mathcal{X}||} \exp(2\pi i(x_0 + jr)k/q) \right|^2 = \frac{1}{q||\mathcal{X}||} \left| \sum_{j=1}^{||\mathcal{X}||} \exp(2\pi ijrk/q) \right|^2$$

(I think it is possible for the sum to start at $j = 0$ and go to $||\mathcal{X}|| - 1$, but it doesn't matter). This probability is largest when the sum basically runs entirely on the real axis; thus the probability is larger as rk/q approaches an integer.

Therefore, when we measured k, there is a very high probability that $r = \lambda q/k$ for some integer λ.

Thus, at the end of this, we have, with high probability, measured a k such that

$$\frac{k}{q} = \frac{\lambda}{r}$$

for some integer λ, where r is the period.

To extract r, we use the continuous fraction expansion described above in Section 4.1.1. We have that $c = k/q = \lambda/r$. We iterate through the denominators d_i that come out of the recurrence and test to see if any of them satisfy $f(x+r) = f(x)$.

Note that we know from Theorem 1 that the period satisfies $r < N$, so we can truncate our continuous fractions expansion if the denominator is ever greater than or equal to N, and then we must start over at step one.

4.3 Finding factors with Shor's algorithm

Method to find factors of a number N. (Note that N must not be even or an integer power of a prime number.)

1. Randomly pick a positive integer $1 < a < N$. If $\gcd(a, N) \neq 1$, then we have found a nontrivial factor. *done*
2. Otherwise, a is coprime with N. Use the quantum period-finding method discussed above to compute the period r of $f(x) = a^x \mod N$. (In my code, to apply $a^x \mod N$ directly may cause an overflow depending on the a and the x. Thus, I write a function that iteratively calculates $a^x \mod N$ based on Theorem 3).
3. If r is odd or $a^{r/2} \equiv -1 \mod N$, restart at step 1. Otherwise, we have sufficient conditions to use Theorem 4. Thus, both $\gcd(a^{r/2} + 1, N)$ and $\gcd(a^{r/2} - 1, N)$ are nontrivial factors of N. *done*

Note that we can compute the gcd's for Shor's algorithm using Theorem 2.

5 Quantum addition

If we initialize our register to be a particular state with probability one, than we can add deterministically by simply applying particular gates. If our register is in a superposition, then the each state will add in a particular way, but the result is random from the measurement.

5.1 Ripple carry adder

This is not exactly a quantum algorithm; I will not explain in detail, as this algorithm is essentially the same as the addition of binary numbers via the classical ripple carry algorithm, the only difference being that AND and XOR gates are renamed to Tofolli and Controlled-Not gates (unitary operators).

Anyone interested should simply add two binary numbers by hand and see how it works and how to carry digits. For the addition of two n bit numbers, the algorithm I implement uses $3n$ qubits, with qubits 0 through $n-1$ storing the first number, n through $2n-1$ storing the second number, $2n$ through $3n-2$ being intermediate bits that store information about carrying over, and the $3n-1$ bit is an overflow (since adding two n bit numbers can result in a $n+1$ bit number).

To see the full algorithm, see the code.

5.2 Adder using the QFT

Consider the numbers x and y represented in binary with n qubits. Begin with the state $|x, y\rangle$. We want to find an operator A that completes the sequence below:

$$|x, y\rangle \xRightarrow[\text{to } x]{\text{QFT}} \frac{1}{\sqrt{2^n}} \sum_k \exp(2\pi i k x / 2^n) \, |k, y\rangle \overset{A}{\Rightarrow} \frac{1}{\sqrt{2^n}} \sum_k \exp(2\pi i k (x+y) / 2^n) \, |k, z\rangle \xRightarrow[\text{to } x]{\text{IQFT}} |(x+y) \mod 2^n, z\rangle$$

where we don't really care what z is. Again, for fear of plagarizing, I refer the reader to page 6 of [2] to see how this operation A can be decomposed into Controlled-Phase gates. To see how it works, one must represent the result of the QFT in terms of decimals in binary. Represent x and y in terms of a string of binary digits. Then the phases in the exponent of the Fourier sum can be easily represented since dividing by 2^n in binary is like dividing by 10^n in base ten.

The full combination of gates can be seen in the code.

Acknowledgments

I love Wikipedia. Quantumplayground.net was also used a few times. [1] is a great resource that I used as well.

References

[1] Isaac Chuang and Michael Nielsen, *Quantum Computation and Quantum Information*, 2000. ISBN: 978-1-107-00217-3.

[2] `https://arxiv.org/pdf/quant-ph/0008033v1.pdf`