8.06 Spring 2015 Lecture Notes

5. Quantum Computing

Aram Harrow

Last updated: May 23, 2016

Quantum computing uses familiar principles of quantum mechanics, but with a different philosophy. The key differences are

- It looks at the information carried by quantum systems, and methods of manipulating it, in ways that are independent of the underlying physical realization (i.e. spins, photons, superconductors, etc.)
- Instead of attempting to describe the world around us, it asks what we can create. As a result, features of quantum mechanics that appear to be limitations, such as Heisenberg's uncertainty principle, can be turned into new capabilities, such as secure quantum key distribution.

The approach is of course analogous to classical computing, where the principles of computing are the same whether your information is stored in magnetic spins (e.g. a hard drive) or electric circuits (e.g. RAM). After the modern notion of a computer was invented in the 1930s by Alan Turing, Alonzo Church, and others, many believed that all computing models (ranging from modern CPUs to DNA computers to clerks working with pencil and paper) were fundamentally equivalent. Specifically, the **Strong Church-Turing thesis** held that any reasonable computing model could simulate any other with modest overhead. Relativity turns out to still be compatible with this hypothesis, but as we will see, quantum mechanics is not.

1 Classical computing

Model a classical computer by a collection of m bits $x_1, \ldots, x_m$ upon which we perform *gates*, each acting on $O(1)$ bits. For example, the NOT gate acts on a single bit, maps 0 to 1, and 1 to 0. The NAND gate means "not-AND" is defined by

x	y	x NAND y
0	0	1
0	1	1
1	0	1
1	1	0

The NAND gate is useful because it is *computationally universal*. Using only NAND gates, we can construct *any* function on n-bit strings.

Here, we can imagine the output of a gate overwriting an existing bit, but it is often simpler to work with *reversible* computers. The NOT gate is reversible. Another example of a reversible gate is the *controlled-NOT* or CNOT, which maps a pair of bits (x, y) to $(x, x \oplus y)$. Here $\oplus$ denotes addition modulo 2.

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

For reversible circuits, computational universality means the ability to implement any invertible function from $\{0,1\}^n$ (the set of n-bit strings) to $\{0,1\}^n$. It turns out that NOT and CNOT gates alone are not computationally universal (proving this is a nice exercise; as a hint, they generate only transformations that are linear mod 2). For computational universality in reversible circuits, we need some 3-bit gate. One convenient one is the Toffoli gate (named after its inventor, Tom Toffoli), which maps (x, y, z) to $(x, y, z \oplus xy)$. If we start with $(x, y, 1)$, then applying the Toffoli gate yields $(x, y, x \text{ NAND } y)$. Since the Toffoli gate can simulate the NAND gate, it is computationally universal.

There are not significant differences between the reversible and irreversible models, but the reversible one will be a more convenient way to discuss quantum computing.

1.1 Complexity

The complexity of a function is the minimum number of gates required to implement it. Since the exact number of gates is not very illuminating, and will in general depend on the details of the computational model, we often care instead about how the complexity scales with the input size n. For example, the time to multiply two n-bit numbers using the straightforward method requires time $O(n^2)$, since this is the number of pairs of bits that have to be considered. The true complexity is lower, since a more clever algorithm is known to multiply numbers in time $O(n \log(n))$. Factoring integers, on the other hand, is not something we know how to do quickly. The naive algorithm to factor an n-bit number x requires checking all primes $\leq \sqrt{x}$, which requires checking $\approx 2^{n/2}/n$ numbers. A much more complicated algorithm is known (the number field sieve) which requires time $O(2^{n^{1/3}})$, and modern cryptosystems are based on the assumption that nothing substantially faster is possible.

In general, we say that a problem can be solved efficiently if we can solve in time polynomial in the input size n, i.e. in time $\leq n^c$ for some constant $c > 0$. So multiplication can be solved efficiently, as well as many other problems, like inverting a matrix, minimizing a convex function, finding the shortest path in a graph, etc. On the other hand, many problems besides factoring are also not known to have any fast algorithms, of which one example is the "taxicab-ripoff problem", namely finding the *longest* path in a graph without repeating any vertex.

The strong Church-Turing essentially states that modern computers (optionally equipped with true random number generators) can simulate any other model of computation with polynomial overhead. In other words "polynomial-time" means the same on any platform.

1.2 The complexity of quantum mechanics

Feynman observed in 1982 that quantum mechanics appeared to violate the strong Church-Turing thesis. Consider the state of n spin-1/2 particles. This state is a unit vector in $(\mathbb{C}^2)^{\otimes n} = \mathbb{C}^{2^n}$, so to describe it requires 2^n complex numbers. Similarly solving the Schrödinger equation on this system appears to require time exponential in n on a classical computer.

Already this suggests that the classical model of computing does not capture everything in the world. But Feynman then asked whether in this case a hypothetic *quantum computer* might be able to do better. On pset 10, you will have the option of exploring a technique by which quantum computers can simulate general quantum systems with overhead that is polynomial in the space and time of the region being simulated.

2 Quantum computers

2.1 Qubits and gates

A quantum computer can be thought of as n spin-1/2 particles whose Hamiltonian is under our control. A single spin-1/2 particle is called a *qubit* and we write its state as $a_0|0\rangle + a_1|1\rangle$ (think of $|0\rangle = |\uparrow\rangle$ and $|1\rangle = |\downarrow\rangle$ if you like, but $|0\rangle = |\text{ground}\rangle$, $|1\rangle = |\text{first excited}\rangle$ works equally well). The state of two qubits can be written as

$$a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle,$$

where we use $|x_1x_2\rangle$ as a shorthand for $|x_1\rangle \otimes |x_2\rangle$. And the state of n qubits can be written as

$$|\psi\rangle = \sum_{\vec{x}\in\{0,1\}^n} a_{\vec{x}}|\vec{x}\rangle.$$

The power of quantum computing comes from the fact that acting on one or two qubits is equivalent to applying a $2^n \times 2^n$ matrix to the state $|\psi\rangle$. Let's see concretely how this works.

If we have a single qubit and can apply any H we like for any time t, we can generate any unitary matrix $U = e^{-iHt}$. (Use units where $\hbar = 1$.) This is because any unitary matrix U can be diagonalized as $U = \sum_{j=1}^{2} e^{i\theta_j}|v_j\rangle\langle v_j|$, and so applying the Hamiltonian $\sum_j -\theta_j|v_j\rangle\langle v_j|$ for time 1 will yield U.

What if we have two qubits? If we apply H to the first qubit then this is equivalent to the two-qubit Hamiltonian $H \otimes I_2$. The resulting unitary is

$$V = e^{-i(H\otimes I)} = e^{-iH} \otimes I = U \otimes I.$$

Similarly if we apply H to the second qubit, then this corresponds to the two-qubit Hamiltonian $I \otimes H$, which generates the unitary operation $I \otimes U$. If we instead apply a two-qubit Hamiltonian, then we can generate any two-qubit (i.e. 4×4) unitary.

This does not scale up to general n-qubit unitaries. Instead we perform these by stringing together sequences of one and two-qubit gates. Some important one-qubit gates (each of which is

both Hermitian and unitary, meaning their eigenvalues are $1, -1$) are

$$X = \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$Y = \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

$$Z = \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{\sqrt{2}} \sum_{x,y \in \{0,1\}} (-1)^{xy} |x\rangle\langle y|$$

The last gate H is known as the Hadamard transform and plays an important role in quantum computing. It has the useful properties that $HXH = Z$ and $HZH = X$.

An important two-qubit is the controlled-NOT gate, which is discussed further on pset 10.

How do these look when acting on n qubits? If we apply a one-qubit (2×2) gate U to the j^{th} qubit, this results in the unitary

$$I_2^{\otimes j-1} \otimes U \otimes I_2^{\otimes n-j}.$$

If we apply a two-qubit gate V to qubits $j, j+1$, then we have the unitary

$$I_2^{\otimes j-1} \otimes V \otimes I_2^{\otimes n-j-1}.$$

There is no reason two-qubit gates cannot be applied to non-consecutive qubits, but it requires more notation, so we will avoid this.

One fact we will use later is that

$$\begin{aligned} H^{\otimes n} &= \frac{1}{\sqrt{2}} \sum_{x_1,y_1 \in \{0,1\}} (-1)^{x_1 y_1} |y_1\rangle\langle x_1| \otimes \frac{1}{\sqrt{2}} \sum_{x_2,y_2 \in \{0,1\}} (-1)^{x_2 y_2} |y_2\rangle\langle x_2| \otimes \cdots \otimes \frac{1}{\sqrt{2}} \sum_{x_n,y_n \in \{0,1\}} (-1)^{x_n y_n} |y_n\rangle\langle x_n| \\ &= \frac{1}{\sqrt{2^n}} \sum_{\vec{x},\vec{y} \in \{0,1\}^n} (-1)^{x_1 y_1 + \ldots + x_n y_n} |\vec{y}\rangle\langle \vec{x}| \\ &= \frac{1}{\sqrt{2^n}} \sum_{\vec{x},\vec{y} \in \{0,1\}^n} (-1)^{\vec{x} \cdot \vec{y}} |\vec{y}\rangle\langle \vec{x}| \end{aligned}$$

2.2 Quantum computing is at least as strong as classical computing

I claim that one and two-qubit gates can implement the quantum Toffoli gate. There is a construction with 6 CNOT gates and many single-qubit gates which I will not describe here. This means that given the ability to do arbitrary one- and two-qubit gates, a quantum computer can perform any *classical* reversible computation with asymptotically constant overhead.

This is good to know, but it is in going beyond classical computing that quantum computers become interesting. On the other hand, we may often want to use classical algorithms as a subroutine.

If $f : \{0,1\}^n \to \{0,1\}^n$ is a reversible (i.e. invertible) function, then the corresponding unitary is

$$U = \sum_{\vec{x} \in \{0,1\}^n} |f(\vec{x})\rangle\langle \vec{x}|.$$

3 Grover's algorithm

Grover's algorithm shows a quantum speedup for the simple problem of *unstructured search*. Suppose that f is a function from $\{0, \ldots, N-1\} \to \{0, 1\}$ and our goal is to find some x for which $f(x) = 1$. How many times do we need to evaluate f to find such an x? For simplicity, assume that there is a single w for which $f(w) = 1$. (w stands for "winner.") Evaluating $f(x)$ for random x will find w in an expected $N/2$ steps.

Remarkably, quantum computers can improve this by a quadratic factor, and can find w using $O(\sqrt{N})$ steps.

The algorithm has two ingredients. First, we need to be able to implement the unitary

$$A = \sum_{0 \le x < N} (-1)^{f(x)} |x\rangle\langle x| = I - 2|w\rangle\langle w|.$$

An explanation of how to do this is on pset 10. The unitary A can be thought of as a reflection about the state $|w\rangle$.

Next, we will need to perform the reflection

$$B = 2|s\rangle\langle s| - I,$$

where $|s\rangle = \frac{1}{\sqrt{N}} \sum_{0 \le x < N} |x\rangle$ is the uniform superposition state. This is closely related to the problem of preparing $|s\rangle$. If $N = 2^n$, then $H^{\otimes n}|0^n\rangle = |s\rangle$. (Here 0^n is the string of n zeroes.) Similarly $B = (H^{\otimes n})(2|0^n\rangle\langle 0^n| - I)(H^{\otimes n})$. Thus the problem of implementing B reduces to that of implementing $I - 2|0\rangle\langle 0|$. This in turn, reduces (again using pset 10) to the problem of detecting when a string is equal to the all-zeroes string, which is an easy classical computation.

Finally we can describe Grover's algorithm. It is

- Start with $|s\rangle$.
- Apply AB T times, where T will be determined later.
- Measure.

The key to analyzing Grover's algorithm is that we never leave the two-dimensional subspace spanned by $\{|w\rangle, |s\rangle\}$. These vectors are linearly independent, but not orthogonal. To construct an orthonormal basis from them, define

$$|v\rangle = \frac{1}{\sqrt{N-1}} \sum_{x \ne w} |x\rangle,$$

so that $|s\rangle = \sqrt{\frac{N-1}{N}}|v\rangle + \frac{1}{\sqrt{N}}|w\rangle$.

In the $\{|v\rangle, |w\rangle\}$ basis we have

$$A = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$B = 2|s\rangle\langle s| - I = 2 \begin{pmatrix} \frac{N-1}{N} & \frac{\sqrt{N-1}}{N} \\ \frac{\sqrt{N-1}}{N} & \frac{1}{N} \end{pmatrix} - I = \begin{pmatrix} 1 - \frac{2}{N} & \frac{2\sqrt{N-1}}{N} \\ \frac{2\sqrt{N-1}}{N} & \frac{2}{N} - 1 \end{pmatrix}$$

$$AB = \begin{pmatrix} 1 - \frac{2}{N} & \frac{2\sqrt{N-1}}{N} \\ -\frac{2\sqrt{N-1}}{N} & 1 - \frac{2}{N} \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$$

If N is large, then $\theta = \sin^{-1}(2\sqrt{N-1}/N) \approx 2/\sqrt{N}$. Thus, after $T = \frac{\pi/2}{\theta} \approx \frac{\pi}{4}\sqrt{N}$ steps, we have rotated from $|s\rangle \approx |v\rangle$ to $\approx |w\rangle$, and measuring will have a high probability of returning the correct answer.

An alternate explanation of this rotation by $O(1/\sqrt{N})$ comes from the fact that we have reflected across two lines which are at an angle of $O(1/\sqrt{N})$ with each other. See the blackboard for a diagram.

4 Simon's algorithm

Grover's algorithm reduced $O(N)$ time to $O(\sqrt{N})$ for a very general problem. Shor's algorithm offers a more dramatic speedup for a more specialized problem: finding the prime factors of a large integer. As described above, the best known classical algorithms for factoring n-bit numbers require time slightly larger than $2^{n^{1/3}}$. By contrast, a quantum computers could factor an n-bit number using Shor's algorithm in time $O(n^3)$.

The key ingredient of Shor's algorithm is *period-finding*, meaning the problem of finding a given the ability to evaluate a function f for which $f(x) = f(x+a)$ for all x. This is straightforward, but takes time, so first I'll explain the algorithm which was one of the key pieces of inspiration for Shor's algorithm.

Let f be a function on $\{0,1\}^n$ with the promise that $f(\vec{x}) = f(\vec{y})$ if and only if $\vec{x} \oplus \vec{y} = \vec{a}$ for some secret $\vec{a} \in \{0,1\}^n$. The goal of the problem is to find $\vec{a}$.

Define $\mathbb{Z}_2 = \{0,1\}$ with $+$ meaning $\oplus$. We can check that $+, \cdot$ satisfy all the usual properties of addition and multiplication, and so $\mathbb{Z}_2^n \cong \{0,1\}^n$ is a vector space. We will see later why this is a useful move.

Consider the following algorithm.

- Start with the uniform superposition
$$H^{\otimes n}|0^n\rangle = \frac{1}{\sqrt{2^n}} \sum_{\vec{x} \in \mathbb{Z}_2^n} |\vec{x}\rangle.$$
- Compute f and store the answer in a second register.
$$\frac{1}{\sqrt{2^n}} \sum_{\vec{x} \in \mathbb{Z}_2^n} |\vec{x}\rangle \otimes |f(\vec{x})\rangle.$$
- Measure the second register. Suppose the answer is α. Then we are left with a superposition over all $\vec{x}$ such that $f(\vec{x}) = \alpha$. By assumption, this state must have the form
$$\frac{|\vec{x}\rangle + |\vec{x} + \vec{a}\rangle}{\sqrt{2}}.$$
- Apply $H^{\otimes n}$ to obtain
$$\frac{1}{\sqrt{2^{n+1}}} \sum_{\vec{y} \in \mathbb{Z}_2^n} ((-1)^{\vec{x}\cdot\vec{y}} + (-1)^{(\vec{x}+\vec{a})\cdot\vec{y}})|\vec{y}\rangle.$$
- Measure, and obtain outcome y with probability
$$\Pr[y] = \frac{1}{2^{n+1}}(1 + (-1)^{\vec{a}\cdot\vec{y}})^2 = \begin{cases} \frac{1}{2^{n-1}} & \text{if } \vec{a}\cdot\vec{y} = 0 \\ 0 & \text{if } \vec{a}\cdot\vec{y} = 1 \end{cases}$$

Thus, this procedure gives us a random vector $\vec{y}$ that is orthogonal to $\vec{a}$. After we have collected $n-1$ linearly independent such $\vec{y}$ we can determine $\vec{a}$ using linear algebra on $\mathbb{Z}_2^n$. After we have collected $k \leq n-2$ vectors $\vec{y}^{(1)}, \ldots, \vec{y}^{(k)}$, the probability that a new $\vec{y}$ is in the span of $\vec{y}^{(1)}, \ldots, \vec{y}^{(k)}$ is $2^{n-k} \leq 1/4$. Thus we need to run this entire procedure $O(n)$ times in order to find $\vec{a}$ with high probability.

By contrast, a classical algorithm would require $\geq 2^{n/2}$ evaluations of f in order to have a non-negligible chance of finding $\vec{a}$.

5 Shor's algorithm

Suppose we'd like to factor a large number N, say 2048 bits long. One approach is to try dividing by all primes up to $\sqrt{N}$. This takes a long time; on the order of 2^{1024} for our example. The best known classical algorithm is the General Number Field Sieve, which runs in time $\exp(\log(N)^{1/3} \cdot \text{poly}(\log\log(N)))$, and has not yet been used on numbers larger than 768 bits. By contrast, Shor's factoring algorithm runs in time $O(\log^3(N))$.

There are three main components to Shor's algorithm.

1. First factoring is reduced to "period-finding" which means finding an unknown period of a black-box function. This step uses number theory and not quantum mechanics.

2. A quantum algorithm for period-finding is given. It is similar to Simon's algorithm but instead of the Hadamard it uses a transform called the Quantum Fourier Transform (QFT).

3. Finally we describe an efficient circuit for the QFT.

We begin with the number theory part.

Euclid's Algorithm. Given two integers y and z you can quickly find the greatest common divisor $\gcd(y, z)$.

Given N choose a random $1 < a < N$ and assume $\gcd(a, N) = 1$. (If not, we have already found a nontrivial factor!) Define $\text{ord}(a)$ to be the least positive integer r such that $a^r \bmod N = 1$. Is there always such an r? Yes, consider the sequence $a, a^2 \bmod N, a^3 \bmod N, \ldots$ This sequence must eventually repeat. If $a^x \equiv a^y \bmod N$ and $y > x$ then $a^{y-x} \equiv 1 \bmod N$. (This is not obvious but can be proved using tools like the Chinese Remainder Theorem.)

Now suppose we have found $r = \text{ord}(a)$. This means that $a^r - 1 = mN$ for some integer m. Suppose that

- r is even, and
- $a^{r/2} + 1$ is not a multiple of N.

(If either of these fails, try another a. It turns out there is a reasonable probability that one of these holds.) Suppose even more concretely that $N = p_1^{k_1} \ldots p_l^{k_l}$. Then

$$(a^{r/2} - 1)(a^{r/2} + 1) = m p_1^{k_1} \ldots p_l^{k_l} \tag{1}$$

and since $a^{r/2} + 1$ is not a multiple of N there must be some p_j that divides $a^{r/2} - 1$. We can now compute $\gcd(a^{r/2} - 1, N)$ using Euclid's algorithm and we will find a nontrivial factor of N.

That's it for the number theory. Now how do we find $\text{ord}(a)$? We use period finding! Define

$$f(x) = a^x \bmod N. \tag{2}$$

This has period r. Indeed

$$f(x+r) = a^{x+r} \bmod N = a^x a^r \bmod N = a^x \bmod N = f(x). \tag{3}$$

The intuition behind period finding is similar to what we did for Simon's algorithm, although now the Fourier transform is instead more like the traditional (i.e. $U(1)$) Fourier transform. Here's how this works. Choose n such that $N^2 \leq 2^n < 2N^2$. We will evaluate $f(x)$ on $x \in \{0, 1, \ldots, 2^n - 1\}$.

Before describing Shor's algorithm we present a subroutine, the *Quantum Fourier Transform.*

$$U_{\mathrm{QFT}}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} e^{2\pi i x y/2^n} |y\rangle. \tag{4}$$

This is a unitary transformation and I claim it can be performed using $O(n^2)$ gates. Let's see how we use it first and then show how to perform it next.

The quantum part of Shor's algorithm consists of these steps:

- Prepare the state $\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes |0\rangle$.
- Calculate $f(x)$ in the second register to obtain

  $$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes |f(x)\rangle.$$

- Discard the second register. (But we just calculated it!) We can think of this as measuring the register and ignoring the answer. The first register will be in a superposition of the form

  $$C(|x_0\rangle + |x_0 + r\rangle + |x_0 + 2r\rangle + \ldots).$$

  for a random choice of x_0. Let's make this more precise and say the state is

  $$\frac{1}{\sqrt{m}} \sum_{k=0}^{m-1} |x_0 + kr\rangle, \tag{5}$$

  with $m \approx 2^n/r$. At this point our superposition has period r. Think of this as a state with period r in time. If we Fourier transform we should get a peak in frequency proportional to $1/r$ (in fact $2^n/r$ for our discretized Fourier transform). Let's now do the math and see that this happens.

- Apply U_{QFT}. This yields

  $$\frac{1}{\sqrt{m2^n}} \sum_{k=0}^{m-1} \sum_{y=0}^{2^n-1} e^{2\pi i x_0 y/2^n} e^{2\pi i k r/2^n} |y\rangle. \tag{6}$$

- Measure y. We find

  $$\Pr[y] = \frac{1}{m2^n} \left| \sum_{k=0}^{m-1} e^{\frac{2\pi i k r y}{2^n}} \right|^2. \tag{7}$$

To get some intuition for (7), suppose $y \approx 2^n j/r$ for some integer j. Then the phase will be ≈ 1 and we will get $\Pr[y] \approx m/2^n = 1/r$. Let's argue this more precisely. Observe that $\sum_{k=0}^{m-1} z^k = (1-z^m)/(1-z)$. Consider $y = 2^n j/r + \delta_j$ for $|\delta_j| \leq 1/2$. Then

$$\Pr[y] = \frac{1}{m2^n} \left| \frac{1 - \exp\left(\frac{2\pi i m r y}{2^n}\right)}{1 - \exp\left(\frac{2\pi i r y}{2^n}\right)} \right|^2 = \frac{1}{m2^n} \left| \frac{1 - \exp\left(\frac{2\pi i m r \delta_j}{2^n}\right)}{1 - \exp\left(\frac{2\pi i r \delta_j}{2^n}\right)} \right|^2 \tag{8}$$

Using $mr \approx 2^n$ and $|1 - e^{i\theta}| = 2\sin(\theta/2)$ we have

$$\Pr[y] \approx \frac{1}{m2^n} \frac{\sin^2(\pi\delta_j)}{\sin^2(\pi\delta_j/m)} \geq \frac{1}{m2^n} \frac{(\pi\delta_j/(\pi/2))^2}{(\pi\delta_j/m)^2} = \frac{4}{\pi^2}\frac{1}{r}. \tag{9}$$

Therefore with probability $\geq 4/\pi^2$ we obtain $j2^n/r$ from which we can extract r.

- Extract r using more number theory. This part is a classical computation so I will not dwell on it. From the last step we obtain $\frac{y}{2^n} = \frac{j}{r} + \frac{\delta_j}{2^n}$ for an unknown integer j and unknown $|\delta_j| \leq 1/2$. Since $r < N$ and $N^2 \leq 2^n$, finding the best rational approximation with denominator $\leq N$ will do the job. This "best rational approximation" can be found by something called the continued fraction expansion, which we illustrate with an example.

$$\frac{49}{300} = \frac{1}{\frac{300}{49}} = \frac{1}{6 + \frac{6}{49}} = \frac{1}{6 + \frac{1}{\frac{49}{6}}} = \frac{1}{6 + \frac{1}{8 + \frac{1}{6}}}$$

  Rational approximations are obtained by truncating this series, e.g. $1/(6 + 1/8) = 8/49 \approx 49/300$.

For the last twenty years there have been many attempts to generalize Shor's algorithm to other groups. In some cases these would break other cryptosystems. It is often possible to find efficient quantum Fourier transforms for other groups but finding analogues of the continued fraction expansion and the other ingredients has been elusive.

Finally let's talk about how to implement $U_{\mathrm{QFT}}^{(n)}$. The superscript denotes the number of qubits. When $n = 1$ this is just our old friend the Hadamard transform.

$$U_{\mathrm{QFT}}^{(1)} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

When $n = 2$ we get fourth roots of unity, namely $\pm i$ in addition to ± 1:

$$U_{\mathrm{QFT}}^{(2)} = \frac{1}{\sqrt{4}} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

In general the matrix elements of $U_{\mathrm{QFT}}^{(n)}$ look like $\exp(2\pi i xy/2^n)$. Let's see how this looks in terms fo the bits of x and y. Write $x = x_0 + 2x_1 + 4x_2 + \ldots + 2^{n-1}x_{n-1}$ and similarly $y = y_0 + \ldots + 2^{n-1}y_{n-1}$. Then

$$xy \bmod 2^n = x_0y_0 + 2(x_1y_0 + x_0y_1) + \ldots + 2^{n-1}(x_0y_{n-1} + \ldots + x_{n-1}y_0).$$

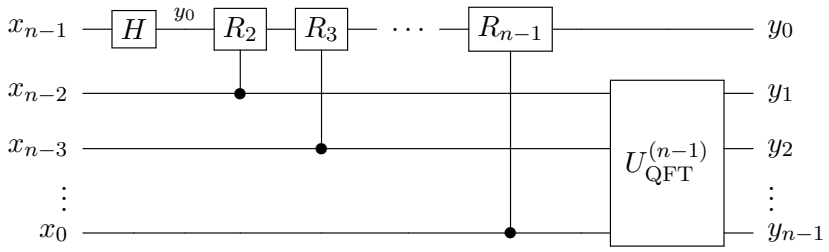

Figure 1: Quantum circuit implementing the quantum Fourier transform.

Write $x = 2^{n-1}x_{n-1} + \bar{x}$ and $y = 2\bar{y} + y_0$. Then

$$\frac{1}{\sqrt{2^n}} e^{\frac{2\pi i x y}{2^n}} = \frac{1}{\sqrt{2^n}} e^{\frac{2\pi i 2^{n-1} x_{n-1} y_0}{2^n}} e^{\frac{2\pi i \bar{x} y_0}{2^n}} \underbrace{e^{\frac{2\pi i 2^{n-1} x_{n-1} 2\bar{y}}{2^n}}}_{1} e^{\frac{2\pi i \bar{x} 2\bar{y}}{2^n}} = \frac{(-1)^{x_{n-1} y_0}}{\sqrt{2}} e^{\frac{2\pi i \bar{x} y_0}{2^n}} \langle \bar{y} | U_{\mathrm{QFT}}^{(n-1)} | \bar{x} \rangle.$$

This suggests a recursive algorithm for $U_{\mathrm{QFT}}^{(n)}$ which we express as a circuit diagram. See Fig. 1. There we have defined the rotation gate

$$R_k \equiv \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}. \tag{10}$$

8.06 Quantum Physics III

Spring 2016