School of Mathematics Spring 2018

QUANTUM COMPUTATION

Lecture notes

Ashley Montanaro, University of Bristol
ashley.montanaro@bristol.ac.uk

Contents

For updates and further materials, see https://people.maths.bris.ac.uk/~csxam/teaching/qc2018/.

Version 1.33 (May 17, 2018).

1 Introduction

Quantum computers are designed to use quantum mechanics to outperform any possible standard, "classical" computer based only on classical physics. In this unit we will introduce the field of quantum computation and study some of the most important ideas in this area. These include quantum computational complexity and the quantum circuit model; the famous algorithms of Shor and Grover for integer factorisation and unstructured search, and the algorithm for simulation of quantum systems; decoherence and quantum error-correction.

1.1 Complementary reading

The Quantum Information Theory unit (MATHM5610) is a prerequisite for this one, and we will aim to follow the notation from that unit where possible. The materials from that unit will therefore be very helpful.

These lecture notes have benefited significantly from the expositions in the following lecture courses, which may be of interest for background reading:

- *Quantum Computation*, Richard Jozsa, University of Cambridge
  `http://www.qi.damtp.cam.ac.uk/node/261`
  The material here on the QFT and Shor's algorithm follows this exposition closely.
- *Quantum Algorithms*, Andrew Childs, University of Waterloo
  `http://www.cs.umd.edu/~amchilds/qa/`
  An excellent resource for more advanced topics than those covered here.
- *Theory of Quantum Information*, John Watrous, University of Waterloo
  `https://cs.uwaterloo.ca/~watrous/LectureNotes.html`
  A particularly useful resource for the theory of quantum channels.

The following books and survey papers may also be useful:

- *Quantum Computation and Quantum Information*, Nielsen and Chuang
  Cambridge University Press, 2001
  The Bible of quantum computing.
- *Classical and Quantum Computation*, Kitaev, Shen and Vyalyi
  American Mathematical Society, 2002
  A more concise introduction to many important topics in quantum computation.
- *Quantum algorithms for algebraic problems*, Childs and van Dam
  Reviews of Modern Physics, 82:1, 2010; `http://arxiv.org/pdf/0812.0380.pdf`
  Covers many other quantum algorithms than those discussed here.
- *Computational Complexity*, Papadimitriou
  Addison-Wesley, 1994
  A comprehensive introduction to classical computational complexity.

1.2 Notation

We write $[n] := \{1, \dots, n\}$ for the integers between 1 and n, and $\mathbb{Z}_n$ for the group of integers modulo n, often just identified with the set $\{0, \dots, n-1\}$. $\lceil x \rceil$, $\lfloor x \rfloor$ and $\lfloor x \rceil$ denote the smallest integer y such that $y \geq x$, the largest integer z such that $z \leq x$, and the closest integer to x, respectively. We use $\binom{n}{k}$ for the binomial coefficient "n choose k", $n!/(k!(n-k)!)$. We say a randomised or quantum algorithm is "bounded-error" if its failure probability is upper-bounded by some constant strictly less than 1/2.

We use standard "computer science style" notation relating to asymptotic complexity:

- $f(n) = O(g(n))$ if there exist real $c > 0$ and integer $n_0 \geq 0$ such that for all $n \geq n_0$, $f(n) \leq c\, g(n)$.
- $f(n) = \Omega(g(n))$ if there exist real $c > 0$ and integer $n_0 \geq 0$ such that for all $n \geq n_0$, $f(n) \geq c\, g(n)$. Clearly, $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$.
- $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

O, Ω and Θ can be viewed as asymptotic, approximate versions of $\leq$, $\geq$ and $=$.

1.3 Change log

- v1.0: first version of notes, corresponding to first parts of 2018 unit.
- v1.1: addition of material on Grover's algorithm, the QFT and Shor's algorithm.
- v1.2: addition of material on phase estimation and quantum simulation.
- v1.3: addition of material on quantum channels and quantum error-correction.
- v1.31: correction of typo in definition of the QFT.
- v1.32: correction of typo in approximate periodicity discussion.
- v1.33: correction of typo in query complexity section.

2 Classical and quantum computational complexity

Computational complexity theory aims to classify different problems in terms of their difficulty, or in other words the resources required in order to solve them. Two of the most important types of resources one might study are *time* (the number of computational steps used by an algorithm solving a problem) and *space* (the amount of additional work space used by the algorithm). Classically, the formal model underpinning the notion of an "algorithm" is the Turing machine. We will not go into details about this here, instead taking the informal approach that the number of steps used by an algorithm corresponds to the number of lines of code executed when running the algorithm, and the space usage is the amount of memory (RAM) used by the algorithm. For much more on the topic, see the book *Computational Complexity* by Papadimitriou, for example.

Rather than looking at the complexity of algorithms for solving one particular instance of a problem, the theory considers asymptotics: given a family of problems, parametrised by an instance size (usually denoted n), we study the resources used by the best possible algorithm for solving that family of problems. Thus the term "problem" is used henceforth as shorthand for "family of problems". A dividing line between efficient and inefficient algorithms is provided by the notion of polynomial-time computation, where an algorithm running in time polynomial in n, i.e. $O(n^c)$ for some fixed c, is considered efficient. For example, consider the following two problems:

- Primality testing: given an integer M expressed as n binary digits, is it a prime number?
- Factorisation: given an integer M expressed as n binary digits, output the prime factors of M.

As the input is of size n, we would like to solve these problems using an algorithm which runs in time $\text{poly}(n)$ (not $\text{poly}(M)$!). No such classical algorithm is known for the factorisation problem; as we will see later, the situation is different for quantum algorithms. However, surprisingly, there is a polynomial-time classical algorithm for the tantalisingly similar problem of primality testing.

An important class of problems is known as decision problems; these are problems that have a yes-no answer. The first of the above problems is a decision problem, while the second is not. But it can be made into a decision problem without changing its underlying complexity significantly:

- Factorisation (decision variant): given integers M and K expressed as n binary digits each, does M have a prime factor smaller than K?

It is clear that, if we can solve the usual "search" variant of the factorisation problem, solving the decision variant is easy. Further, solving the decision variant allows us to solve the search variant of the problem using binary search. Given an integer M whose prime factors we would like to determine, and an algorithm which solves the decision variant of the factorisation problem, we can use $O(\log M) = O(n)$ evaluations of this algorithm with different values of K to find the smallest prime factor F of M. (First we try $K = \lceil M/2 \rceil$, then either $K = \lceil M/4 \rceil$ or $K = \lceil 3M/4 \rceil$, etc.) The other factors can be found by dividing M by F and repeating. This is a simple example of a *reduction*: conversion of one problem into another.

A natural way to compare the complexity of problems is via the notion of complexity classes, where a complexity class is simply a set of problems. Some important classical complexity classes are:

- P: the class of decision problems which can be solved in polynomial time by a classical computer.

- NP: the class of decision problems such that, if the answer is "yes", there is a proof of this fact which can be verified in polynomial time by a classical computer.
- PSPACE: the class of decision problems which can be solved in polynomial space by a classical computer.

Primality testing is in P, although this was shown for the first time only in 2002. The decision variant of factorisation is in NP, because given a claimed prime factor of M smaller than K, it can be easily checked whether the claim is correct. However, factorisation is not known to be in P. Every problem in P is automatically in NP, because the verifier can simply ignore any claimed proof and solve the problem directly. In addition, any problem in NP is automatically in PSPACE, because one can loop over all polynomial-length proofs in polynomial space in order to determine whether the answer to a problem instance should be "yes" or "no". Thus we have P$\subseteq$NP$\subseteq$PSPACE.

A problem is said to be NP-complete if it is in NP, and every other problem in NP reduces to it in polynomial time. So NP-complete problems are, informally, the "hardest" problems in NP. These include many problems of practical importance in areas such as timetabling, resource allocation, and optimisation. One simple example is the Subset Sum problem. An instance of this problem is a sequence of integers $x_1, \dots, x_n$; our task, given such a sequence, is to determine whether there is a subset of the integers which sums to 0. Given such a subset, we can easily check that it sums to 0; however, finding such a subset seems to require checking exponentially many subsets.

NP stands for "nondeterministic polynomial-time", *not* "non-polynomial time". In fact, it is currently unknown whether every problem in NP can be solved in polynomial time, i.e. whether P=NP. This is the famous P vs. NP question; resolving it would win you a prize of \$1M from the Clay Mathematics Institute.

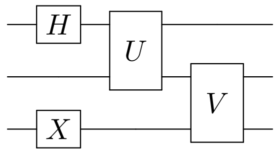
2.1 Quantum computational complexity

The basic framework in which quantum computation operates is as follows. We have a system of n qubits, and choose a basis $\{|0\rangle, |1\rangle\}$ for each qubit. By taking tensor products, this gives a basis of the form $\{|x\rangle : x \in \{0,1\}^n\}$ for the whole system, where for conciseness we write $|x\rangle$ for $|x_1\rangle|x_2\rangle \dots |x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \dots \otimes |x_n\rangle$. This basis is called the *computational basis.* A quantum computation is the application of some unitary operator U to some initial state (usually $|0\rangle^{\otimes n} = |0^n\rangle$, which we often just write as $|0\rangle$), followed by a measurement of k of the qubits in the computational basis, giving some outcome $y \in \{0,1\}^k$. This outcome is then the output of the computation. If the state of a quantum computer is $\sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$ for some coefficients α_x, and we measure all of the qubits, the output is x with probability $|\alpha_x|^2$.

How are we to measure resource usage by a quantum algorithm running on a quantum computer? One framework within which to do this is the *quantum circuit model.* Although in quantum mechanics evolutions of a quantum system are described by unitary operators, not all unitary operators are equally easy to implement physically. We might imagine that, in a real quantum computing experiment, the operations that we can actually perform in the lab are small, "local" ones on just a few qubits at a time. We can build up more complicated unitary operators as products of these small, elementary operations.

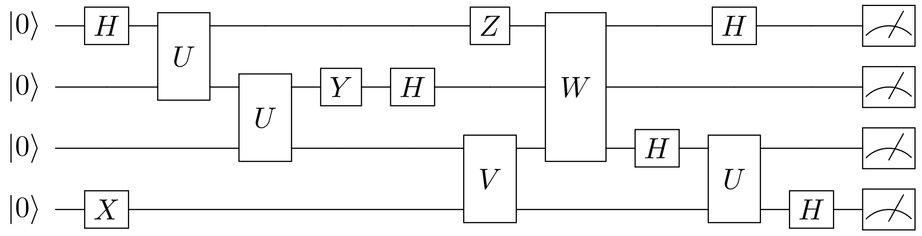
Intuitively, a quantum computation running for T steps and using space S corresponds to a unitary operation on S qubits (i.e. acting on $\mathbb{C}^{2^S}$) expressed as a product of T elementary operations picked from some family $\mathcal{F}$. Each elementary operation is assumed to take time $O(1)$ and act nontrivially on $O(1)$ qubits. That is, if U is one such elementary operation, we assume that it can be written as $U = U' \otimes I$, where U' acts on k qubits, for some small constant k (usually, $k \leq 3$). The nontrivial parts U' of such operations are called *quantum gates*, by analogy with logic gates in classical electronic circuits. The set of allowed quantum gates will depend on our physical architecture. However, it turns out that most "reasonable" sets of gates on $O(1)$ qubits are universal, in the sense that any unitary operation on S qubits can be approximately decomposed as a product of these basic operations, acting on different qubits. A sequence of quantum gates is known as a quantum circuit.

A quantum circuit can be drawn as a diagram by associating each qubit with a horizontal "wire", and drawing each gate as a box across the wires corresponding to the qubits on which it acts. This is easiest to illustrate with an example: the circuit



corresponds to the unitary operator $(I \otimes V)(U \otimes I)(H \otimes I \otimes X)$ on 3 qubits. Beware that a circuit is read left to right, with the starting input state on the far left, but the corresponding unitary operators act right to left! For convenience, in the diagram we have drawn multi-qubit gates as only acting on nearest-neighbour qubits, but this is not an essential restriction of the model.

The quantum circuit picture also allows us to represent initial state preparation, and final measurement of the qubits in the computational basis, as shown in this more complicated example:

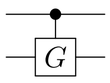


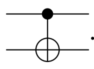
If we prefer, we can allow intermediate measurements during the circuit; this turns out not to change the power of the model.

Some special gates turn out to be particularly useful. You are already familiar with the Hadamard gate H, which is expressed as the matrix $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ (with respect to the computational basis), and the gates X, Y, Z corresponding to the Pauli operators. We can think of the X gate as implementing a NOT operation, as $X|0\rangle = |1\rangle$, $X|1\rangle = |0\rangle$. Another useful type of gate will be the "controlled-G" gates. For any gate G, the corresponding controlled-G gate CG uses an extra qubit to control whether the gate is applied or not. That is,

$$CG|0\rangle|\psi\rangle = |0\rangle|\psi\rangle, \quad CG|1\rangle|\psi\rangle = |1\rangle G|\psi\rangle.$$

In a circuit diagram, this is denoted using a filled circle on the control line:



A particularly useful such gate is controlled-NOT (CNOT), denoted . Written as a matrix with respect to the computational basis,

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

For any fixed gate set $\mathcal{F}$, some large unitary matrices cannot be decomposed efficiently in terms of gates from $\mathcal{F}$, in the sense that to write them as a product of gates from $\mathcal{F}$ requires exponentially many such gates. For a rough way of seeing this, consider the problem of producing an arbitrary quantum state of n qubits $\sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$, in the special case where each coefficient $\alpha_x \in \{\pm 1/2^{n/2}\}$. There are 2^{2^n} such states. Any circuit on n qubits made up of T gates, each acting on k qubits, picked from a gate set of size G can be described by one of

$$\left(G \binom{n}{k} \right)^T = O\left((Gn^k)^T \right) = O\left(2^{T \log(Gn^k)} \right)$$

different sequences of gates, so for $k, G = O(1)$ we need $T \log n = \Omega(2^n)$ to be able to produce $2^{\Omega(2^n)}$ different unitary operators, and hence 2^{2^n} different states. A similar argument still works if we allow approximate computation or continuous gate sets.

In general, just as in the classical world, we look for efficient quantum circuits which use poly(n) qubits and poly(n) gates to solve a problem on an input of size n. The class of decision problems which can be solved by a quantum computer, in time polynomial in the input size, with probability of failure at most 1/3, is known as BQP ("bounded-error quantum polynomial-time"). This class encapsulates the notion of efficient quantum computation. The failure probability bound of 1/3 is essentially arbitrary; it can be reduced to an arbitrarily small constant by repetition and taking the majority vote.

Observe that in the quantum circuit picture we can perform multiple operations in parallel, so we in fact have two possible ways to measure "time" complexity: circuit size (number of gates) and circuit depth (number of time steps to execute all the gates). But these can only differ by a factor of $O(S)$, where S is the number of qubits on which the circuit operates.

2.2 Classical and reversible circuits

Any classical computation which maps a bit-string (element of $\{0,1\}^n$) to another bit-string (element of $\{0,1\}^m$) can be broken down into a sequence of logical operations, each of which acts on a small number of bits (e.g. AND, OR and NOT gates). Such a sequence is called a (classical) circuit[1]. As a first step in understanding the power of quantum computers, we would like to show that any classical circuit can be implemented as a quantum circuit, implying that quantum computation is at least as powerful as classical computation.

But there is a difficulty: in quantum mechanics, if we wish the state of our system to remain pure, the evolution that we apply has to be unitary, and hence reversible. Some classical logical operations (such as AND, written $\wedge$) are not reversible. However, reversible variants of these can be developed using the following trick. If we wish to compute an arbitrary classical operation $f : \{0,1\}^n \to \{0,1\}^m$ reversibly, we attach a so-called "ancilla" register of m bits, each orignally set to 0, and modify f to give a new operation $f' : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n \times \{0,1\}^m$ which

[1]Where "circuit" is understood in the sense of "electronic circuit".

performs the map

$$f'(x,y) = (x, y \oplus f(x)),$$

where $\oplus$ is bitwise XOR, i.e. addition modulo 2 (so each bit of $y \oplus f(x)$ is the sum mod 2 of the corresponding bits of y and $f(x)$). Then if we input $y = 0^m$, we get $(x, f(x))$, from which we can extract our desired output $f(x)$. If we perform f' twice, we get $(x, y \oplus f(x) \oplus f(x)) = (x, y)$. So f is reversible. And any reversible function that maps bit-strings to bit-strings corresponds to a permutation matrix, which is unitary, so can be implemented as a sequence of quantum gates. If we combine many gates of this form to compute a function $f : \{0,1\}^n \to \{0,1\}$, say, we will finish with an output of the form $(\text{junk}, x, f(x))$. If we wish to remove the junk, we can simply copy the output $f(x)$ onto a fresh ancilla bit in state 0 by applying a CNOT gate, and then repeat all the previous gates in reverse. As each is its own inverse, the final state of the computation is $(0, x, f(x))$.

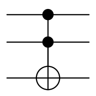
To obtain universal deterministic classical computation, it is sufficient to be able to implement the NOT and AND gates. The NOT gate is immediately reversible. Applying the above construction to AND we get the map $(x_1, x_2, y) \mapsto (x_1, x_2, y \oplus (x_1 \wedge x_2))$ for $x_1, x_2, y \in \{0,1\}$. The unitary operator which implements this is then simply the map

$$|x_1\rangle|x_2\rangle|y\rangle \mapsto |x_1\rangle|x_2\rangle|y \oplus (x_1 \wedge x_2)\rangle.$$

Written as a matrix with respect to the computational basis this is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix},$$

an operation known as the Toffoli gate. In a circuit diagram, the Toffoli gate is written as "controlled-controlled-NOT", i.e.



Randomised classical computation can also be embedded in a quantum circuit. Imagine we have a classical computation which makes use of some random bits, each of which is 0 or 1 with equal probability. We can simulate this by applying a Hadamard gate to $|0\rangle$ to produce the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Then we can either measure this qubit immediately to obtain a uniformly random bit, or if we prefer, apply classical gates to it and then measure it at the end of the computation; the result is the same either way.

It is known that the Toffoli gate, together with the Hadamard gate, are even sufficient for universal *quantum* computation. That is, any quantum computation whatsoever can be approximately represented as a circuit of Toffoli and Hadamard gates. Another representative universal set of quantum gates is $\{H, X, \text{CNOT}, T\}$, where $T = \left(\begin{smallmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{smallmatrix}\right)$. It turns out that almost any non-trivial set of gates is universal in this sense; therefore, we generally do not worry about the details of the gate set being used.

2.3 Query complexity

While time complexity is a practically important measure of the complexity of algorithms, it suffers from the difficulty that it is very hard to prove lower bounds on it, and that technical details can sometimes obscure the key features of an algorithm. One way to sidestep this is to use a model which is less realistic, but cleaner and more mathematically tractable: the model of query complexity.

In this model, we assume we have access to an *oracle*, or "black box", to which we can pass *queries*, and which returns answers to our queries. Our goal is to determine some property of the oracle using the minimal number of queries. On a classical computer, we can think of the oracle as a function $f : \{0,1\}^n \to \{0,1\}^m$. We pass in inputs $x \in \{0,1\}^n$, and receive outputs $f(x) \in \{0,1\}^m$. How does this fit into physical reality? We imagine we are given access to the oracle either as a physical device which we cannot open and look inside, or as a circuit which we can see, but for which it might be difficult to compute some property of the circuit. For example, even given a description of a circuit computing some function $f : \{0,1\}^n \to \{0,1\}$, it might be hard to find an input x such that $f(x) = 1$. Sometimes it is more natural to think of an oracle function f as a memory storing 2^n strings of m bits each, where we can retrieve an arbitrary string at the cost of one query.

We can give a quantum computer access to a oracle using the generic reversible computation construction discussed in the previous section. That is, instead of having a function $f : \{0,1\}^n \to \{0,1\}^m$, we produce a unitary operator O_f which performs the map[1]

$$O_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle.$$

O_f is sometimes known as the *bit oracle*. If $m = 1$, so f returns one bit, it would also make sense to consider an oracle U_f which does not use an ancilla, but instead flips the phase of an input state $|x\rangle$ by applying the map

$$U_f|x\rangle = (-1)^{f(x)}|x\rangle.$$

This variant is thus sometimes known as the *phase oracle*. Given access to a bit oracle, we can simulate a phase oracle by attaching an ancilla qubit in the state $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$:

$$O_f|x\rangle\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}(|x\rangle|f(x)\rangle - |x\rangle|f(x) \oplus 1\rangle) = (-1)^{f(x)}|x\rangle\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

Note that the ancilla qubit is left unchanged by this operation, which is called the phase kickback trick. Also note that the effect of the phase oracle is not observable if we apply it to just one computational basis state $|x\rangle$, but only if we apply it to a superposition:

$$\sum_{x \in \{0,1\}^n} \alpha_x |x\rangle \mapsto \sum_{x \in \{0,1\}^n} (-1)^{f(x)} \alpha_x |x\rangle.$$

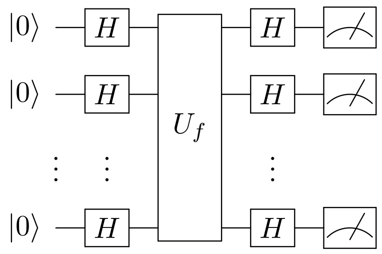
Importantly, note that to implement the oracles O_f and U_f we do not need to understand any more about the inner workings of f than we do classically. That is, if we are given a classical circuit computing f, we can follow a purely mechanical construction to create quantum circuits implementing O_f and U_f. This is useful because f itself may have quite complicated behaviour, even if it is expressible as a small circuit, and we may not be able to understand its behaviour completely.

[1]Note that the notation used here is different to Quantum Information Theory.

2.4 The Deutsch-Jozsa algorithm as a quantum circuit

Recall that the *Deutsch-Jozsa algorithm* can distinguish between balanced and constant functions $f : \{0,1\}^n \to \{0,1\}$ with one use of the oracle U_f, whereas an exact classical algorithm for solving the same problem would require exponentially many (in n) queries to f. That is, if we are promised either that f is constant or that $|\{x : f(x) = 0\}| = |\{x : f(x) = 1\}| = 2^{n-1}$, we can determine which is the case with one quantum query.

We now verify that this algorithm can be implemented as an efficient quantum circuit on n qubits. Indeed, the circuit is very simple:



The evolution of the input state throughout the circuit is

$$|0\rangle^{\otimes n} \mapsto \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)} \left(\frac{1}{\sqrt{2^n}} \sum_{y\in\{0,1\}^n} (-1)^{x\cdot y}|y\rangle \right)$$

where $x \cdot y = \sum_{i=1}^n x_i y_i$. To see the last step, observe that

$$H^{\otimes n}|x\rangle = (H|x_1\rangle) \otimes (H|x_2\rangle) \otimes \cdots \otimes (H|x_n\rangle),$$

and that $H|x_i\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^{x_i}|1\rangle)$. So

$$\begin{aligned} H^{\otimes n}|x\rangle &= \frac{1}{\sqrt{2^n}}(|0\rangle + (-1)^{x_1}|1\rangle)(|0\rangle + (-1)^{x_2}|1\rangle)\dots(|0\rangle + (-1)^{x_n}|1\rangle) \\ &= \frac{1}{\sqrt{2^n}} \sum_{y\in\{0,1\}^n} \prod_{i:y_i=1} (-1)^{x_i}|y\rangle = \frac{1}{\sqrt{2^n}} \sum_{y\in\{0,1\}^n} (-1)^{\sum_{i:y_i=1} x_i}|y\rangle = \frac{1}{\sqrt{2^n}} \sum_{y\in\{0,1\}^n} (-1)^{x\cdot y}|y\rangle. \end{aligned}$$

We can rewrite the final state in the algorithm as

$$\sum_{y\in\{0,1\}^n} \frac{1}{2^n} \left(\sum_{x\in\{0,1\}^n} (-1)^{f(x)+x\cdot y} \right) |y\rangle.$$

Consider the case $y = 0^n$. Then $\frac{1}{2^n}\sum_{x\in\{0,1\}^n}(-1)^{f(x)+x\cdot y} = \frac{1}{2^n}\sum_{x\in\{0,1\}^n}(-1)^{f(x)}$. If f is constant, this quantity evaluates to ± 1; if f is balanced, the sum evaluates to 0. So in the former case, the outcome 0^n is obtained by the final measurement with certainty; in the latter case, an outcome other than 0^n is obtained with certainty. So we can distinguish between the cases with certainty.

The delicate cancellation effects occurring in this algorithm are a common feature of quantum algorithms; we will see them again later in Shor's algorithm.

3 Grover's algorithm

A simple example of a problem that fits into the query complexity model is unstructured search on a set of N elements for a unique marked element. In this problem we are given access to a function $f : [N] \to \{0, 1\}$ with the promise that $f(x_0) = 1$ for a unique "marked" element x_0. Our task is to output x_0.

It is intuitively clear that the unstructured search problem should require about N queries to be solved (classically!). We can formalise this as the following proposition:

Proposition 3.1. *Let $\mathcal{A}$ be a classical algorithm which solves the unstructured search problem on a set of N elements with failure probability $\delta < 1/2$. Then $\mathcal{A}$ makes $\Omega(N)$ queries in the worst case.*

Proof sketch. We can think of any classical algorithm $\mathcal{A}$ as choosing in advance, either deterministically or randomly, a sequence S of distinct indices $x_1, \dots, x_N$ to query, and then querying them one by one until the marked element x_0 is found. Imagine an adversary chooses x_0 uniformly at random. Then, on average, x_0 will appear at position about $N/2$ in the sequence S. As for a random choice of x_0 the algorithm makes $\Omega(N)$ queries on average, the average number of queries made in the worst case must also be $\Omega(N)$. □

In the quantum setting, we will see that the unstructured search problem can be solved with significantly fewer queries.

Theorem 3.2 (Grover '97)**.** *There is a quantum algorithm which solves the unstructured search problem using $O(\sqrt{N})$ queries.*

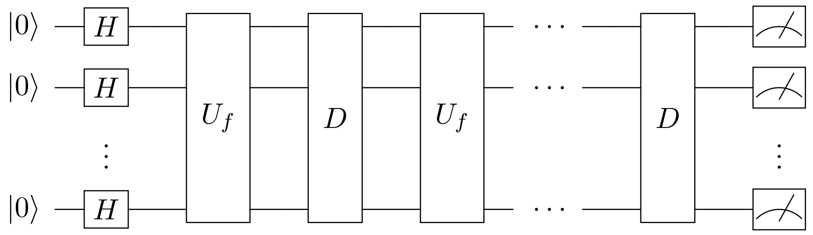
For simplicity, assume that $N = 2^n$ for some integer n (this is not an essential restriction). In this case, we can associate each element of $[N]$ with an n-bit string. Then Grover's algorithm is described in Box 1.

We are given access to $f : \{0, 1\}^n \to \{0, 1\}$ with the promise that $f(x_0) = 1$ for a unique element x_0. We use a quantum circuit on n qubits with initial state $|0\rangle^{\otimes n}$. Let H denote the Hadamard gate, and let U_0 denote the n-qubit operation which inverts the phase of $|0^n\rangle$: $U_0|0^n\rangle = -|0^n\rangle$, $U_0|x\rangle = |x\rangle$ for $x \neq 0^n$.

1. Apply $H^{\otimes n}$.
2. Repeat the following operations T times, for some T to be determined later:
   (a) Apply U_f.
   (b) Apply $D := -H^{\otimes n} U_0 H^{\otimes n}$.
3. Measure all the qubits and output the result.

Box 1: Grover's algorithm

The overall unitary operation performed is thus $(-H^{\otimes n} U_0 H^{\otimes n} U_f)^T H^{\otimes n}$. (Incidentally, note that the minus sign in front of D can actually be omitted without affecting the correctness of the algorithm, but it is helpful for the analysis.) In circuit diagram form, Grover's algorithm looks like this:



It may be far from clear initially why this algorithm works, or indeed whether it does work. To describe the algorithm, we introduce unitary operators $I_{|\psi\rangle}$ and $R_{|\psi\rangle}$, where $|\psi\rangle$ is an arbitrary state. These are defined as follows:

$$I_{|\psi\rangle} := I - 2|\psi\rangle\langle\psi|, \quad R_{|\psi\rangle} := -I_{|\psi\rangle} = 2|\psi\rangle\langle\psi| - I,$$

where I is the identity. $I_{|\psi\rangle}$ can be seen as an "inversion about $|\psi\rangle$" operation, while $R_{|\psi\rangle}$ can be seen as a "reflection about $|\psi\rangle$" operation. An arbitrary state $|\phi\rangle$ can be expanded as

$$|\phi\rangle = \alpha|\psi\rangle + \beta|\psi^\perp\rangle$$

for some α and β, and some state $|\psi^\perp\rangle$ such that $\langle\psi|\psi^\perp\rangle = 0$. Then

$$I_{|\psi\rangle}|\phi\rangle = -\alpha|\psi\rangle + \beta|\psi^\perp\rangle,$$

so $I_{|\psi\rangle}$ has flipped the phase of the component corresponding to $|\psi\rangle$, and left the component orthogonal to $|\psi\rangle$ unchanged. $R_{|\psi\rangle}$ has the opposite effect. Observe that, in the unstructured search problem with marked element x_0, $U_f = I_{|x_0\rangle}$. Further observe that

$$H^{\otimes n} U_0 H^{\otimes n} = H^{\otimes n}(I - 2|0^n\rangle\langle 0^n|)H^{\otimes n} = I - 2|+\rangle\langle +| = I_{|+\rangle},$$

where $|+\rangle = \frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|x\rangle$, so $D = -I_{|+\rangle}$. By moving the minus sign, the algorithm can equally well be thought of as alternating the operations $-I_{|x_0\rangle}$ and $I_{|+\rangle}$, or equivalently $R_{|x_0\rangle}$ and $-R_{|+\rangle}$.

We have the following claims:

1. For any states $|\psi\rangle$, $|\phi\rangle$, and any state $|\xi\rangle$ in the 2d plane spanned by $|\psi\rangle$ and $|\phi\rangle$, the states $R_{|\psi\rangle}|\xi\rangle$ and $R_{|\phi\rangle}|\xi\rangle$ remain in this 2d plane.

   This is immediate from geometric arguments, but one can also calculate explicitly:

$$\begin{aligned} R_{|\psi\rangle}(\alpha|\psi\rangle + \beta|\phi\rangle) &= R_{|\psi\rangle}(\alpha|\psi\rangle + \beta(\gamma|\psi\rangle + \delta|\psi^\perp\rangle)) = (\alpha+\beta\gamma)|\psi\rangle - \beta\delta|\psi^\perp\rangle \\ &= (\alpha + 2\beta\gamma)|\psi\rangle - \beta(\gamma|\psi\rangle + \delta|\psi^\perp\rangle) = (\alpha+2\beta\gamma)|\psi\rangle - \beta|\phi\rangle. \end{aligned}$$

2. Within the 2d plane spanned by orthogonal states $|\psi\rangle$, $|\psi^\perp\rangle$, $I_{|\psi\rangle} = -R_{|\psi\rangle} = R_{|\psi^\perp\rangle}$.

   Again, one can calculate explicitly that

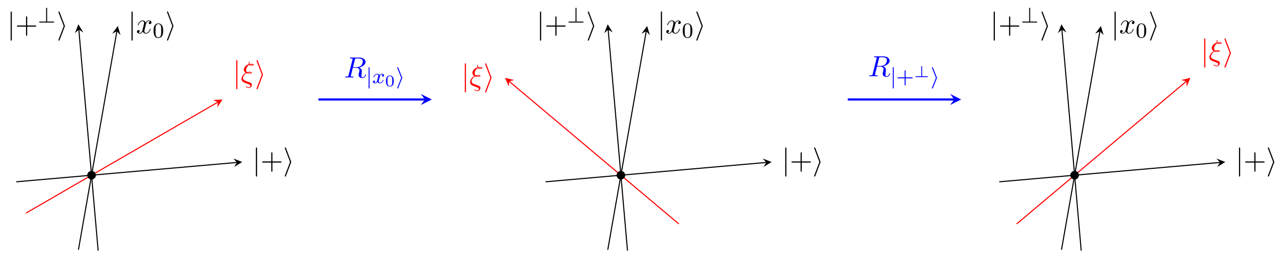
$$-R_{|\psi\rangle}(\alpha|\psi\rangle + \beta|\psi^\perp\rangle) = -\alpha|\psi\rangle + \beta|\psi^\perp\rangle = R_{|\psi^\perp\rangle}(\alpha|\psi\rangle + \beta|\psi^\perp\rangle).$$

3. If $|\xi\rangle$ is within the 2d plane spanned by $|\psi\rangle$, $|\psi^\perp\rangle$,

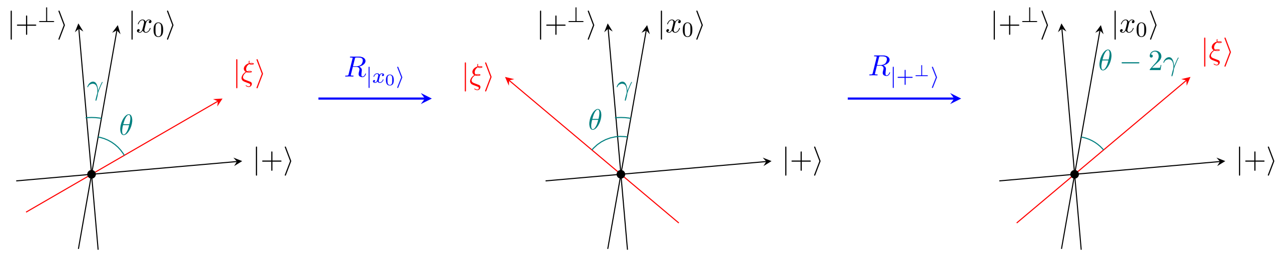
$$R_{|\psi\rangle}|\xi\rangle = \langle\psi|\xi\rangle|\psi\rangle - \langle\psi^\perp|\xi\rangle|\psi^\perp\rangle.$$

   This is just a straightforward calculation.

Combining these claims, we see that each step of Grover's algorithm consists of two reflections in the plane spanned by $|+\rangle$, $|x_0\rangle$: a reflection about $|x_0\rangle$ followed by a reflection about $|+^\perp\rangle$, a state orthogonal to $|+\rangle$ within this plane. We can illustrate this with the following diagram, demonstrating the effect of these operations on an arbitrary state $|\xi\rangle$ within this 2d plane:



We see that $|\xi\rangle$ has moved closer to $|x_0\rangle$. In fact, geometrically speaking, the composition of two reflections is a rotation! If the angle between $|\xi\rangle$ and $|x_0\rangle$ is θ, and the angle between $|x_0\rangle$ and $|+^\perp\rangle$ is γ, composing these two reflections rotates $|\xi\rangle$ in the direction of $|+^\perp\rangle$ by an angle of $2\theta - 2(\theta - \gamma) = 2\gamma$. This is proven by picture in the following diagram but could also be shown using the representation of rotations and reflections by 2d matrices.



Repeating the Grover iteration continues to rotate $|\xi\rangle$ within this plane by angle 2γ. We stop when we are as close to $|x_0\rangle$ as possible. We start with $|\xi\rangle = |+\rangle$, so the initial angle between $|\xi\rangle$ and $|x_0\rangle$ is $\pi/2 - \gamma$. We can calculate what γ is by using the formula $\cos\gamma = \langle x_0|+^\perp\rangle$, so $\sin\gamma = \langle x_0|+\rangle = 1/\sqrt{N}$. As $\sin x \approx x$ for small x, we expect the number of iterations required to move from an angle of $\pi/2 - \gamma$ down to an angle of 0 to be about $(\pi/4)\sqrt{N}$. One can calculate this more precisely: after T iterations, the angle between $|\xi\rangle$ and $|x_0\rangle$ is

$$\gamma_T := \pi/2 - (2T+1)\arcsin(1/\sqrt{N}),$$

so the probability of obtaining the outcome x_0 when we measure is precisely

$$|\langle\xi|x_0\rangle|^2 = \cos^2(\gamma_T) = \sin^2((2T+1)\arcsin(1/\sqrt{N})). \tag{1}$$

Maximising this by taking T to be the integer nearest to

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

we learn x_0 with probability $1 - O(1/N)$ using $O(\sqrt{N})$ queries. (The above expression uses $\arcsin x = x + O(x^3)$ for small x.) Figure 2 illustrates the success probabilities for $N = 100$.

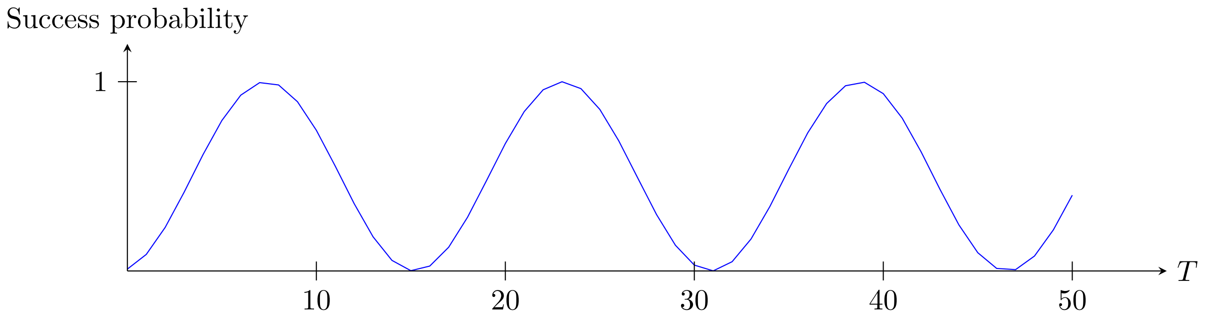


Figure 2: Success probabilities of Grover's algorithm for $N = 100$.

We see that, as the number of uses of the Grover iterate increases past $(\pi/4)\sqrt{N}$, the success probability starts to decrease.

A particularly nice case, where we can determine an exact solution, is $N = 4$. Here we have $\arcsin(1/2) = \pi/6$, so if we plug in $T = 1$ to Eqn. (1), the probability of getting the outcome x_0 is $\sin^2(\pi/2) = 1$; so we get the right answer with certainty after only one query.

We have calculated the query complexity of Grover's algorithm; what is the time complexity? As well as the calls to U_f, we need to implement the operation D. But this can be done efficiently: D consists of two layers of n Hadamard gates and an operation which flips the phase if the input is not all 0's. This operation – which is based on computing the bitwise OR of n bits – can be implented using $O(\log n)$ layers of classical gates. So the overhead is $O(n)$ gates per iteration, and depth only $O(\log n)$. This is minor compared with the number of iterations, which is $\Theta(2^{n/2})$.

3.1 Multiple marked elements

Grover's algorithm can also be used when there are $M > 1$ marked elements. In this setting, the operator U_f inverts the phase of input elements $x \in S$, for some unknown subset $S \subseteq [N]$, where $|S| = M$. U_f is still related to an inversion operator, but now an inversion about an M-dimensional subspace:

$$U_f = I - 2\Pi_S,$$

where $\Pi_S = \sum_{x \in S} |x\rangle\langle x|$. If we define the state $|S\rangle := \frac{1}{\sqrt{M}} \sum_{x \in S} |x\rangle$, we see that

$$\begin{aligned} I_{|S\rangle}|+\rangle &= (I - 2|S\rangle\langle S|)|+\rangle = |+\rangle - 2\left(\frac{1}{M}\sum_{x,y\in S}|x\rangle\langle y|\right)\left(\frac{1}{\sqrt{N}}\sum_{x\in\{0,1\}^n}|x\rangle\right) \\ &= |+\rangle - \frac{2}{\sqrt{N}}\sum_{x\in S}|x\rangle = (I - 2\Pi_S)|+\rangle = U_f|+\rangle \end{aligned}$$

and similarly

$$I_{|S\rangle}|S\rangle = -|S\rangle = (I - 2\Pi_S)|S\rangle = U_f|S\rangle.$$

That is, the U_f operation behaves like an inversion-about-$|S\rangle$ operator for any states in the subspace spanned by $|+\rangle$ and $|S\rangle$. The whole of the previous analysis goes through, except that now the angle γ moved at each step satisfies $\sin\gamma = \langle S|+\rangle = \sqrt{M/N}$. Thus after T iterations we have

$$|\langle \xi|S\rangle|^2 = \cos^2(\gamma_T) = \sin^2((2T+1)\arcsin(\sqrt{M/N})).$$

By a similar argument to before we can pick $T \approx (\pi/4)\sqrt{N/M}$ to obtain overlap with $|S\rangle$ close to 1. When we measure at the end of the algorithm, we get an element of the subset S (and in fact a uniformly random such element) with probability $|\langle\xi|S\rangle|^2$. In particular, observe that when $M = N/4$, we again measure an element of S with certainty using only one query.

What if we do not know the number of marked elements in advance? The following simple trick can deal with this. First run the algorithm assuming there is 1 marked element; if it fails, try again assuming there are 2 marked elements; then 4, 8, etc. The total number of queries used is roughly

$$\sum_{k=0}^{\log_2 N} \frac{\pi}{4}\sqrt{\frac{N}{2^k}} = \frac{\pi}{4}\sqrt{N}\sum_{k=0}^{\log N} 2^{-k/2} = O(\sqrt{N}).$$

If the number of marked elements is M', at least one of the iterations must choose a guess M for M' such that $M'/2 \leq M \leq 2M'$. This corresponds to a value of T which is within a factor of about $\sqrt{2}$ of the optimal value $T' \approx (\pi/4)\sqrt{N/M'}$. Then, as $(2T'+1)\arcsin(\sqrt{M'/N}) = \pi/2 + O(\sqrt{M'/N})$,

$$\begin{aligned}\sin^2((2T+1)\arcsin(\sqrt{M'/N})) &= \sin^2\left(\frac{2T+1}{2T'+1}(2T'+1)\arcsin(\sqrt{M'/N})\right)\\ &= \sin^2\left(\frac{2T+1}{2T'+1}(\pi/2 + O(\sqrt{M'/N}))\right),\end{aligned}$$

which is lower-bounded by a strictly positive constant if M is small with respect to N. Repeating the whole algorithm $O(1)$ times, and checking each time whether the returned element is marked, allows us to achieve an arbitrarily high success probability.

This algorithm might still have a high probability of failing in the case where $M = \Omega(N)$. To find a marked element in this case we can just sample $O(1)$ random values of $f(x)$ classically; we will find a marked element with high probability.

3.2 Problems in NP and "database search"

Grover's algorithm is often presented as a way of searching an unstructured database, or a database which is not structured in a way that is useful to us; for example, trying to search by phone number in a phone book ordered by name. However, the primary use of Grover's algorithm (at least initially) is likely not to be searching physical databases, but instead searching for solutions to computational problems.

Grover's algorithm gives a quadratic quantum speedup over classical exhaustive search for any problem in NP. This is because we can choose the oracle operation f to be the classical checking circuit which takes an input a claimed solution, and outputs 1 if the solution is correct, and 0 otherwise. If there are N possible solutions to the problem, Grover's algorithm lets us find a solution using only $O(\sqrt{N})$ checks. Note that this does not immediately imply that Grover's algorithm is better than *any* classical algorithm; in some cases, there could be a more efficient classical algorithm based on using the structure of the problem.

But could we also use Grover's algorithm to search a real database? This would rely on the use of a "quantum RAM" which allowed elements of the memory to be efficiently queried in superposition. In principle, there do not seem any fundamental reasons why such a memory could not be constructed. However, in practice building a quantum RAM is likely to be challenging.

3.3 Amplitude amplification

The basic idea behind Grover's algorithm can be generalised remarkably far, to an algorithm for finding solutions to any problem using a heuristic. This algorithm is known as *amplitude amplification*.

Imagine we have $N = 2^n$ possible solutions, of which a subset S are "good", and we would like to find a good solution. As well as having access to a "checking" algorithm f as before, where $f(x) = 1$ if and only if x is marked, we now have access to a "guessing" algorithm $\mathcal{A}$, which has the job of producing potential solutions to the problem. It performs the map

$$\mathcal{A}|0^n\rangle = \sum_{x\in\{0,1\}^n} \alpha_x|x\rangle$$

for some coefficients $\{\alpha_x\}$. So, if we were to apply $\mathcal{A}$ and then measure, the probability that we would obtain a good solution is

$$p := \sum_{x\in S} |\alpha_x|^2;$$

we think of $\mathcal{A}$ as a heuristic which tries to output a good solution. We can use f to check whether a claimed solution is actually good. If we repeated Algorithm $\mathcal{A}$ until we got a good solution, the expected number of trials we would need is $\Theta(1/p)$.

We now describe the amplitude amplification algorithm.

We are given access to $\mathcal{A}$ and U_f as above.

1. Apply $\mathcal{A}$ to the starting state $|0\rangle^{\otimes n}$.
2. Repeat the following operations T times, for some T to be determined:
   (a) Apply U_f.
   (b) Apply $-\mathcal{A}U_0\mathcal{A}^{-1}$.
3. Measure all the qubits and output the result.

Box 3: Amplitude amplification

Note that this is exactly the same as Grover's algorithm, except that we have replaced the $H^{\otimes n}$ operations with $\mathcal{A}$ or $\mathcal{A}^{-1}$. Write

$$|\psi\rangle = \mathcal{A}|0^n\rangle, \quad |G\rangle = \frac{\Pi_S|\psi\rangle}{\|\Pi_S|\psi\rangle\|},$$

where again $\Pi_S = \sum_{x\in S}|x\rangle\langle x|$. We now repeat the analysis of the previous section, except that we replace $|+\rangle$ with $|\psi\rangle$ and $|S\rangle$ with $|G\rangle$. We observe that everything goes through just as before! The first operation applied is equivalent to $I_{|G\rangle}$, and the second is equivalent to $-I_{|\psi\rangle}$. We start with the state $|\psi\rangle$ and rotate towards $|G\rangle$. The angle γ moved at each step now satisfies

$$\sin\gamma = \langle\psi|G\rangle = \|\Pi_S|\psi\rangle\| = \sqrt{p},$$

so the number of iterations required to move from $|\psi\rangle$ to $|G\rangle$ is $O(1/\sqrt{p})$ – a quadratic improvement.

Finally observe that we can generalise one step further, by replacing the algorithm U_f with inversion about an arbitrary subspace, rather than a subspace defined in terms of computational basis vectors. This allows us to use amplitude amplification to drive amplitude towards an arbitrary subspace, or indeed to create an arbitrary quantum state, given the ability to reflect about that state.

7	3	4	2	9	7	3	4	2	9	7	3	4	2	9	7	3	4	2	9

Figure 4: A periodic sequence, with period 5, which is one-to-one on each period.

4 The Quantum Fourier Transform and periodicity

We now introduce an important unitary transformation which is used in a number of different contexts in quantum information theory: the quantum Fourier transform (QFT) over $\mathbb{Z}_N$, the integers modulo N. This can be seen as a generalisation of the familiar Hadamard gate. The QFT is the map

$$Q_N|x\rangle = \frac{1}{\sqrt{N}} \sum_{y \in \mathbb{Z}_N} \omega_N^{xy} |y\rangle,$$

where $\omega_N := e^{2\pi i/N}$, and xy is just the product of the two numbers x and y, thought of as integers. We sometimes omit the subscript N where there is no ambiguity. Some examples of the QFT in small dimension, with respect to the computational basis:

$$Q_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad Q_3 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 & 1 & 1 \\ 1 & e^{2\pi i/3} & e^{-2\pi i/3} \\ 1 & e^{-2\pi i/3} & e^{2\pi i/3} \end{pmatrix}, \quad Q_4 = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}.$$

To see that the QFT is unitary, we calculate the inner product of rows x and z, which equals

$$\frac{1}{N} \sum_{y \in \mathbb{Z}_N} (\omega_N^{xy})^* \omega_N^{zy} = \frac{1}{N} \sum_{y \in \mathbb{Z}_N} \omega_N^{(z-x)y}.$$

To compute this sum, we use the formula for the sum of a geometric series:

$$\sum_{k=0}^{r-1} x^k = \begin{cases} \frac{1-x^r}{1-x} & \text{if } x \neq 1 \\ r & \text{if } x = 1 \end{cases}, \tag{2}$$

implying that the inner product is equal to 1 if $z = x$, and $\frac{1-\omega_N^{(z-x)N}}{1-\omega_N^{z-x}}$ otherwise. But as $\omega_N^N = 1$, the inner product is 0 if $z \neq x$. More generally, for any integer j,

$$\frac{1}{N} \sum_{y \in \mathbb{Z}_N} \omega_N^{jy} = \begin{cases} 0 & \text{if } j \not\equiv 0 \bmod N \\ 1 & \text{if } j \equiv 0 \bmod N \end{cases}, \tag{3}$$

a fact which will be useful later.

The QFT is exactly the same transformation as the Discrete Fourier Transform (DFT) used for classical computation and signal processing, up to the nonstandard normalisation of $1/\sqrt{N}$.

4.1 Periodicity determination

One of the most important applications of the QFT is determining the period of a periodic function. Imagine we are given access to an oracle function $f : \mathbb{Z}_N \to \mathbb{Z}_M$, for some integers N and M, such that:

- f is **periodic**: there exists r such that r divides N and $f(x+r) = f(x)$ for all $x \in \mathbb{Z}_N$;
- f is **one-to-one** on each period: for all pairs (x, y) such that $|x - y| < r$, $f(x) \neq f(y)$.

Our task is to determine r.

The periodicity determination algorithm is presented in Box 5.

We are given access to a periodic function f with period r, which is one-to-one on each period. We start with the state $|0\rangle|0\rangle$, where the first register is dimension N, and the second dimension M.

1. Apply Q_N to the first register.
2. Apply O_f to the two registers.
3. Measure the second register.
4. Apply Q_N to the first register.
5. Measure the first register; let the answer be k.
6. Simplify the fraction k/N as far as possible and return the denominator.

Box 5: Periodicity determination

The initial sequence of operations which occur during the algorithm is:

$$|0\rangle|0\rangle \overset{1}{\mapsto} \frac{1}{\sqrt{N}} \sum_{x \in \mathbb{Z}_N} |x\rangle|0\rangle \overset{2}{\mapsto} \frac{1}{\sqrt{N}} \sum_{x \in \mathbb{Z}_N} |x\rangle|f(x)\rangle.$$

When the second register is measured, we receive an answer, say z. By the periodic and one-to-one properties of f, all input values $x \in \mathbb{Z}_N$ for which $f(x) = z$ are of the form $x_0 + jr$ for some x_0 and integer j. The state therefore collapses to something of the form

$$\sqrt{\frac{r}{N}} \sum_{j=0}^{N/r-1} |x_0 + jr\rangle.$$

After we apply the QFT, we get the state

$$\frac{\sqrt{r}}{N} \sum_{j=0}^{N/r-1} \left(\sum_{y \in \mathbb{Z}_N} \omega_N^{y(x_0+jr)} |y\rangle \right) = \frac{\sqrt{r}}{N} \sum_{y \in \mathbb{Z}_N} \omega_N^{yx_0} \left(\sum_{j=0}^{N/r-1} \omega_N^{jry} \right) |y\rangle.$$

Observe that, as r divides N, $\omega_N^r = e^{2\pi i (r/N)} = \omega_{N/r}$. This state is thus equivalent to

$$\frac{\sqrt{r}}{N} \sum_{y \in \mathbb{Z}_N} \omega_N^{yx_0} \left(\sum_{j=0}^{N/r-1} \omega_{N/r}^{jy} \right) |y\rangle.$$

By Eqn. (3), the sum over j is 0 unless $y \equiv 0 \bmod N/r$, or in other words if $y = \ell N/r$ for some integer ℓ. So we can rewrite this state as

$$\frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} \omega_N^{\ell x_0 N/r} |\ell N/r\rangle.$$

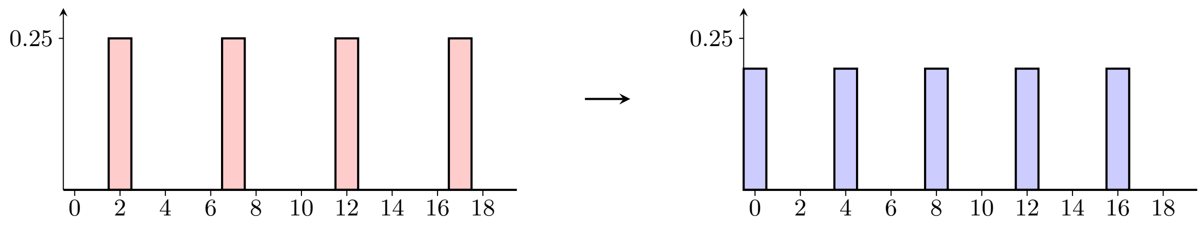


Figure 6: Periodicity determination as above with $N = 20$, $r = 5$. First diagram illustrates the probabilities of measurement outcomes after step 3 (for one possible measurement result for the second register), second diagram illustrates probabilities after step 5.

When we perform the final measurement, we receive an outcome $k = \ell_0 N/r$, for some ℓ_0 picked uniformly at random from $0, \dots, r-1$. We know that

$$k = \frac{\ell_0 N}{r}, \qquad \text{so } \frac{k}{N} = \frac{\ell_0}{r}.$$

In this equation, we know N and k and would like to determine r. If it happened that ℓ_0 were coprime to r, we could cancel the fraction on the left-hand side and output the denominator. What is the probability that we are lucky in this way?

Fact 4.1. *Fix an positive integer a and pick b uniformly at random from the integers between 0 and a. Then the probability that b is coprime to a is $\Omega(1/\log\log a)$.*

Thus, if we repeat the whole procedure $O(\log\log r) = O(\log\log N)$ times, we are quite likely to find the period r. Why? If we have a probabilistic procedure which succeeds with probability p, the probability that it fails every time over R repetitions is exactly

$$(1-p)^R \leq e^{-pR},$$

so it suffices to take $R = O(1/p)$ to achieve, say, 99% success probability. Each time the algorithm returns a claimed period, we can check whether it is really a period of the function using two additional queries. Each use of the quantum algorithm therefore makes 3 queries to O_f, so the whole algorithm makes $O(\log\log N)$ queries in total. In terms of time complexity, the most complicated classical processing required is the elementary arithmetic in step 6, which can be implemented (via Euclid's algorithm, which we will not discuss further here) using $\text{poly}(\log N)$ arithmetic operations. However, we have not yet shown that we can implement the QFT Q_N efficiently.

4.2 Efficient implementation of the QFT

We will show here how to implement Q_N efficiently – i.e. using a circuit of size $O(\text{poly}\log N)$ – in the case where N is a power of 2. (In fact, the QFT can also be implemented (approximately) efficiently when N is not a power of 2.) The efficient implementation is based on the same ideas as the classical Fast Fourier Transform (FFT). To begin with, we observe that the output of the QFT, when applied to a computational basis state, has an efficient description as a product state.

Assume that $N = 2^n$ for some integer n, and represent each $y \in \mathbb{Z}_N$ by the n-bit string

$(y_0, y_1, \dots, y_{n-1})$, where $y = y_0 + 2y_1 + 4y_2 + \dots + 2^{n-1}y_{n-1}$. Then

$$\begin{aligned} Q_N|x\rangle &= \frac{1}{2^{n/2}} \sum_{y \in \mathbb{Z}_{2^n}} \omega_{2^n}^{xy} |y\rangle \\ &= \frac{1}{2^{n/2}} \sum_{y \in \{0,1\}^n} \omega_{2^n}^{x(\sum_{j=0}^{n-1} 2^j y_j)} |y_{n-1}\rangle|y_{n-2}\rangle \dots |y_0\rangle \\ &= \left(\frac{1}{\sqrt{2}} \sum_{y_{n-1} \in \{0,1\}} \omega_{2^n}^{2^{n-1}xy_{n-1}} |y_{n-1}\rangle \right) \left(\frac{1}{\sqrt{2}} \sum_{y_{n-2} \in \{0,1\}} \omega_{2^n}^{2^{n-2}xy_{n-2}} |y_{n-2}\rangle \right) \dots \left(\frac{1}{\sqrt{2}} \sum_{y_0 \in \{0,1\}} \omega_{2^n}^{xy_0} |y_0\rangle \right) \\ &= \bigotimes_{j=1}^{n} \left(\frac{1}{\sqrt{2}} \sum_{y_{n-j} \in \{0,1\}} \omega_{2^j}^{xy_{n-j}} |y_{n-j}\rangle \right). \end{aligned}$$

Because $xy_{n-j} \equiv 0 \bmod 2^j$ when x is an integer multiple of 2^j, we see that the j'th qubit of the output only depends on the j bits $x_0, \dots, x_{j-1}$. We can write this another way, as

$$Q_N|x\rangle = \frac{1}{2^{n/2}} \left(|0\rangle + e^{2\pi i(.x_0)}|1\rangle \right) \left(|0\rangle + e^{2\pi i(.x_1x_0)}|1\rangle \right) \dots \left(|0\rangle + e^{2\pi i(.x_{n-1}\dots x_0)}|1\rangle \right),$$

where the notation $(.x_{j-1}\dots x_0)$ is used for the binary fraction

$$\frac{x_{j-1}}{2} + \frac{x_{j-2}}{4} + \dots + \frac{x_0}{2^j}.$$

So we see that the first qubit of the output depends on only the last qubit of the input, the second qubit depends on the last two, etc. We can utilise this structure by building up the output state in reverse order. The last stage of the circuit creates the correct state for the first qubit, which is then not used again; the last but one stage creates the correct state for the second qubit, etc. To produce the correct state for each qubit, we can use the gates H and R_d, where

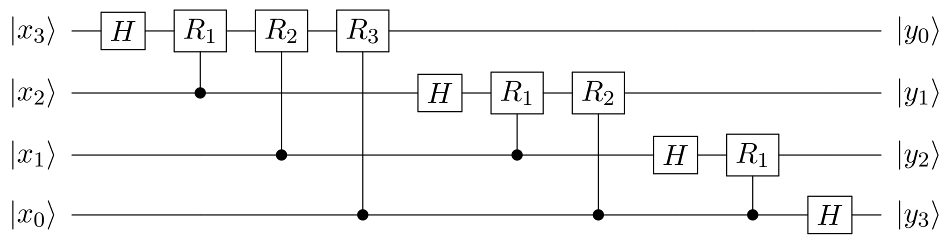
$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad R_d = \begin{pmatrix} 1 & 0 \\ 0 & e^{\pi i/2^d} \end{pmatrix}.$$

Here we are assuming that we have access to R_d gates for arbitrary d; as briefly discussed in Section 2, this is not essential as any universal gate set will allow us to approximately implement these gates. Observe that the Hadamard gate can be written as the map

$$H|x\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i(.x)}|1\rangle \right)$$

for $x \in \{0,1\}$. We can use this to start building up a binary fraction in the phase of the basis state $|1\rangle$. Applying a R_d gate to this state will add $1/2^{d+1}$ to this binary fraction. To apply R_d conditional on the bits of x, we will use controlled-R_d gates.

The easiest way to illustrate this process is with an example. The overall circuit for the QFT on 4 qubits can be depicted as



Observe that the output state is backwards, i.e. the qubits appear in reverse order. They can be returned to the original order, if desired, using swap gates. How many gates in total are used in the circuit? The j'th stage of the circuit, for $j = 1, \ldots, n$, uses one Hadamard gate and $n - j$ R_d gates. Thus the overall number of gates used is $O(n^2)$; $n(n-1)/2$ R_d gates and n Hadamard gates, then n additional swap gates, if used. This is $O(\log^2 N)$, so we have indeed obtained an efficient circuit.

This complexity can be improved further, to $O(n \log n)$, if we are content with an approximate version of the QFT. The observation which implies this is that many of the operations in the circuit are R_d gates for large values of d, which do not affect the output significantly. Indeed, it turns out that there is a constant C such that if we omit the gates R_d with $d \geq C \log n$, for any input state $|x\rangle$ the output is close to the input up to an error of at most $1/\operatorname{poly}(n)$. The modified circuit uses $O(\log n)$ gates at each stage, so $O(n \log n)$ in total.

5 Integer factorisation

The main application of periodicity determination is Shor's quantum algorithm for integer factorisation. Given an n-digit integer N as input, this algorithm outputs a non-trivial factor of N (or that N is prime, if this is the case) with success probability $1-\epsilon$, for any $\epsilon > 0$, in time $O(n^3)$. The best classical algorithm known (the general number field sieve) runs in time $e^{O(n^{1/3} \log^{2/3} n)}$. In fact, this is a heuristic bound and this algorithm's worst-case runtime has not been rigorously determined; the best proven bound is somewhat higher. Shor's algorithm thus achieves a super-polynomial improvement. This result might appear only of mathematical interest, were it not for the fact that the very widely-used RSA public-key cryptosystem relies on the hardness of integer factorisation. Shor's efficient factorisation algorithm implies that this cryptosystem is insecure against attack by a large quantum computer. Although no large-scale quantum computer yet exists, this result has already had major implications for cryptography, as national security agencies start to move away from the RSA cryptosystem.

Unfortunately (?), proving correctness of Shor's algorithm requires going through a number of technical details. First we need to show that factoring reduces to a periodicity problem – though in this case an *approximate* periodicity problem. This part uses only classical number theory. Then we need to show that periodicity can still be determined even in the setting where the input function is only approximately periodic. This part uses the theory of continued fractions.

5.1 From factoring to periodicity

The basic skeleton of the quantum factorisation algorithm is given in Box 7. It is based on two "magic" subroutines. The first is a classical algorithm for computing the greatest common divisor (gcd) of two integers. This can be achieved efficiently using Euclid's algorithm. The second ingredient is an algorithm for computing the order of an integer a modulo N, i.e. the smallest integer r such that $a^r \equiv 1 \bmod N$; this is where we will use periodicity determination. As long as a and N are coprime, such an integer p exists:

Fact 5.1 (Euler's theorem)**.** *If a and N are coprime then there exists r such that $a^r \equiv 1 \bmod N$.*

```
Let $N$ denote the integer to be factorised. Assume that $N$ is not even or a power of a prime.

    1. Choose $1 < a < N$ uniformly at random.
    2. Compute $b = \gcd(a, N)$. If $b > 1$ output $b$ and stop.
    3. Determine the order $r$ of $a$ modulo $N$. If $r$ is odd, the algorithm has failed; terminate.
    4. Compute $s = \gcd(a^{r/2} - 1, N)$. If $s = 1$, the algorithm has failed; terminate.
    5. Output $s$.
```

Box 7: Integer factorisation algorithm (overview)

We start by showing that this algorithm does work, assuming that the subroutines used all work correctly. If a is coprime to N, there exists r such that $a^r \equiv 1 \bmod N$. If, further, such an r

is even, we can factor

$$a^r - 1 = (a^{r/2} + 1)(a^{r/2} - 1) \equiv 0 \bmod N.$$

So N divides the product $(a^{r/2} + 1)(a^{r/2} - 1)$. If neither term in the product is a multiple of N itself, N must divide partly into each of them, so if we computed $\gcd(a^{r/2} \pm 1, N)$, we would obtain a factor of N. Because r is the *smallest* integer x such that $a^x \equiv 1 \bmod N$, we know that $a^{r/2} - 1$ is not divisible by N. We also need that $a^{r/2} + 1$ is not divisible by N. This, and r being even, turn out to occur with quite high probability:

Fact 5.2. *Let N be odd and not a power of a prime. If $1 < a < N$ is chosen uniformly at random with $\gcd(a, N) = 1$, then $\Pr[r$ is even and $a^{r/2} \not\equiv -1 \bmod N] \geq 1/2$.*

The algorithm thus succeeds with probability at least $1/2$. If it fails, we simply repeat the whole process. After K repetitions, we achieve a failure probability of at most $1/2^K$. Even if the order-finding procedure has some small probability of error (which will turn out to be the case), we can check whether the algorithm's output s is correct by attempting to divide N by s.

We assumed throughout that N is not even or a power of a prime. If N is even, we simply output 2 as a factor. To deal with the case that $N = p^\ell$ for some prime p and some integer $\ell > 0$, we observe that we can efficiently compute the roots $N^{1/k}$, for $k = 2, \dots, \log_2 N$. If any of these is an integer, we have found a factor of N. Finally, what about if N is itself prime? In this case the algorithm will fail every time. We can therefore output "prime" after a suitable number of failures.

Example 5.3. *Consider $N = 15$. Imagine we choose $a = 7$ at random. Then $\gcd(7, 15) = 1$. The function $f(x) = 7^x \bmod 15$ takes values $1, 7, 4, 13, 1, \dots$ for $x \geq 0$. So $r = 4$ and we have $(7^2 + 1)(7^2 - 1) \equiv 0 \bmod 15$. The greatest common divisor of $7^2 - 1 = 48$ and 15 is 3, which is indeed a factor of 15.*

It remains to show how to implement step 3. Consider the function $f : \mathbb{Z} \to \mathbb{Z}_N$ defined by

$$f(x) = a^x \bmod N.$$

We have $f(x + y) = f(x)f(y)$ and, by Euler's theorem, $f(r) = 1$. So $f(x + r) = f(x)f(r) = f(x)$ for all x, i.e. f is periodic with period r. Since r is the smallest integer such that $f(r) = 1$, we also have that f is one-to-one on each period. However, although this function is periodic on the whole domain $\mathbb{Z}$, we will need to truncate it to a finite size. If we knew what the period was, we could choose this size to make the function periodic again, but of course we do not know this in advance. This will lead to the function becoming no longer exactly periodic, but just approximately periodic.

5.2 Approximate periodicity

We restrict the function $f(x) = a^x \bmod N$ to the set $x \in \{0, \dots, M - 1\}$, where $M = 2^m$ is the smallest power of 2 greater than N^2 (we will see the reasons behind this choice later). Write $M = Br + b$ for $0 \leq b < r$, where $B = \lfloor M/r \rfloor$. That is, the function is periodic up to the last period, which is truncated and contains only b elements, rather than r. We apply the steps of the periodicity-determination algorithm to f as in Section 4.1. That is, we first construct the state

$$\frac{1}{\sqrt{M}} \sum_{x=0}^{M-1} |x\rangle|f(x)\rangle,$$

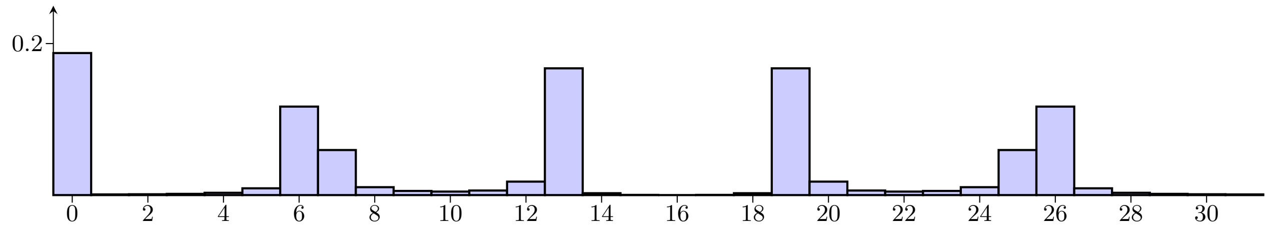


Figure 8: The probabilities of different measurement outcomes for a function with period 5, with $M = 32$. Note the peaks around multiples of $32/5 = 6.4$.

then measure the second register to receive an answer $z = f(x_0)$ for some x_0. The state of the first register then becomes

$$|\psi\rangle = \frac{1}{\sqrt{A}} \sum_{j=0}^{A-1} |x_0 + jr\rangle,$$

where $A = B + 1$ if $x_0 < b$, and $A = B$ if $x_0 \geq b$. Write

$$Q_M|\psi\rangle = \sum_{y=0}^{M-1} \alpha_y |y\rangle$$

for the resulting state when we apply the QFT to $|\psi\rangle$. By direct calculation we have

$$\alpha_y = \frac{1}{\sqrt{MA}} \sum_{j=0}^{A-1} \omega_M^{y(x_0+jr)} = \frac{\omega_M^{yx_0}}{\sqrt{MA}} \sum_{j=0}^{A-1} \left(\omega_M^{yr}\right)^j .$$

Using the formula (2) for the sum of a geometric series, the sum evaluates to

$$\frac{1 - \omega_M^{yrA}}{1 - \omega_M^{yr}}$$

if $yr \not\equiv 0 \bmod M$, and evaluates to A otherwise. Previously, in the case of exact periodicity determination, we had $A = B = M/r$, so the corresponding sum evaluated to 0 unless $yr \equiv 0 \bmod M$, i.e. unless y was a multiple of M/r. Now we aim to show that, when we measure, we get an outcome y which is *close* to a multiple of the non-integer value M/r with high probability. This situation is illustrated in Figure 8.

When we measure, the probability of obtaining outcome y is

$$\Pr[y] = \frac{1}{MA} \left| \frac{1 - \omega_M^{yrA}}{1 - \omega_M^{yr}} \right|^2 = \frac{1}{MA} \left| \frac{1 - e^{2\pi i yrA/M}}{1 - e^{2\pi i yr/M}} \right|^2 = \frac{\sin^2(\pi yrA/M)}{MA \sin^2(\pi yr/M)}.$$

To see the third equality, note that $|1 - e^{i\theta}| = |e^{i\theta/2} - e^{-i\theta/2}| = 2|\sin(\theta/2)|$ for any real θ. Now consider values y of the form $y = \lfloor \ell M/r \rceil$ for some integer ℓ. We call such integers "good". We can write any good integer as $y = \ell M/r + \epsilon$ for some small ϵ such that $|\epsilon| \leq 1/2$. Indeed, we have the slightly stronger bound (which we will need) that $|\epsilon| \leq 1/2 - 1/r$. This holds because (a) $\ell M/r$ is an integer divided by r, so the distance from the closest integer is an multiple of $1/r$; (b) $r < N$ and $M > N^2$ is a power of 2, so any factors of 2 in the denominator of the fraction $\ell M/r$ can be cancelled and we cannot have $|\epsilon| = 1/2$.

Then

$$\Pr[y] = \frac{\sin^2(\pi(\ell M/r+\epsilon)rA/M)}{MA\sin^2(\pi(\ell M/r+\epsilon)r/M)} = \frac{\sin^2(\ell A\pi + \epsilon rA\pi/M)}{MA\sin^2(\ell\pi+\epsilon r\pi/M)} = \frac{\sin^2(\epsilon rA\pi/M)}{MA\sin^2(\epsilon r\pi/M)}$$

by periodicity of the function $|\sin\theta|$. We now claim (see (4) in Box 9) that the following pair of inequalities hold for any θ in the range $0 \le \theta \le \pi/2$:

$$(2/\pi)\theta \le \sin\theta \le \theta.$$

Assuming these inequalities, we have

$$\Pr[y] \ge \frac{(2\epsilon rA/M)^2}{MA(\epsilon r\pi/M)^2} = \frac{4A}{\pi^2 M} \ge \frac{4}{\pi^2 M}\left(\frac{M}{r}-1\right) = \frac{4}{\pi^2 r} - O(1/N^2).$$

Note that the lower bound on the numerator is valid because $|\epsilon| rA\pi/M \le \pi/2$, which was a consequence of $|\epsilon| \le 1/2 - 1/r$:

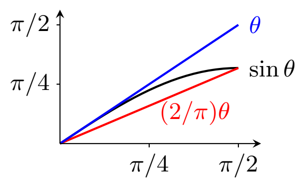
$$\frac{|\epsilon|rA\pi}{M} \le \frac{\left(\frac{1}{2}-\frac{1}{r}\right) r \left(\frac{M}{r}+1\right)\pi}{M} = \frac{\pi}{2}\left(1-\frac{2}{r}\right)\left(1+\frac{r}{M}\right) \le \frac{\pi}{2}\left(1-\frac{2}{r}\right)\left(1+\frac{2}{r}\right) = \frac{\pi}{2}\left(1-\frac{4}{r^2}\right) < \frac{\pi}{2},$$

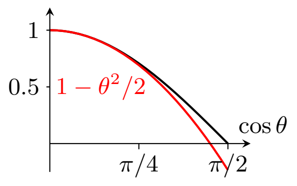
where the second inequality holds because $r^2 \le N^2 \le 2M$. Therefore, as there are r "good" integers y of the form $\lfloor \ell M/r \rceil$, the probability of obtaining at least one of them is at least $4/\pi^2 - O(1/N)$.

The following trigonometric inequalities will be useful for the analysis of various quantum algorithms. We state them without proof, instead referring to the figure below.

$$(2/\pi)\theta \le \sin\theta \le \theta \text{ for } 0 \le \theta \le \pi/2; \quad (4)$$

$$\cos\theta \ge 1 - \theta^2/2 \text{ for all } \theta. \quad (5)$$





Box 9: Trigonometric inequalities

5.3 Learning r from an approximate period

It remains to extract r from an integer y of the form $y = \lfloor \ell M/r \rceil$. Divide y by M to obtain a rational number z such that

$$\left|\frac{\ell}{r} - z\right| < \frac{1}{2M} < \frac{1}{2N^2}.$$

We would like to find the fraction ℓ/r from z. We first claim that there is at most one fraction of the form ℓ'/r' with $r' < N$ satisfying the above bound. To prove this, imagine there were two such fractions ℓ'/r', ℓ''/r''. Then

$$\left|\frac{\ell'}{r'} - \frac{\ell''}{r''}\right| = \frac{|\ell' r'' - r'\ell''|}{r'r''} \ge \frac{1}{r'r''} > \frac{1}{N^2}.$$

But, as ℓ'/r', ℓ''/r'' are each within $1/(2N^2)$ of z, they must be at most distance $1/N^2$ apart, so we have a contradiction.

We have seen that it suffices to find any fraction ℓ'/r' such that $r' < N$ to learn ℓ/r. To do this, we use the theory of continued fractions. The continued fraction expansion (CFE) of z is an expression of the form

$$z = \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \dots}}},$$

where the a_i are positive integers. To find the integers a_i, we start by writing

$$z = \frac{1}{z'},$$

where $z' = a_1 + b$ for some integer a_1 and some $b < 1$; then repeating this process on b. Note that, for any rational z, this expansion must terminate after some number of iterations C. One can show that in fact, for any rational $z = s/t$ where s and t are m-bit integers, $C = O(m)$. Once we have calculated this expansion, if we truncate it after some smaller number of steps, we obtain an approximation to z. These approximations are called convergents of the CFE.

Example 5.4. *The continued fraction expansion of* $z = 31/64$ *is*

$$\frac{31}{64} = \frac{1}{2 + \frac{1}{15 + \frac{1}{2}}}.$$

The convergents are

$$\frac{1}{2}, \quad \frac{1}{2 + \frac{1}{15}} = \frac{15}{31}.$$

Fact 5.5. *Any fraction* p/q *with* $|p/q - z| < 1/(2q^2)$ *will appear as one of the convergents of the CFE of* z.

Therefore, if we carry out the continued fraction expansion of z, we are guaranteed to find ℓ/r, as the unique fraction close enough to z.

Example 5.6. *Imagine we want to factor* $N = 21$. *We set* $M = 512 > 21^2$ *and choose* $a = 10$ *at random. The order of a mod 21 is 6. So we would expect the measurement outcomes we receive to be close to multiples of* $512/6 = 85\frac{1}{3}$. *Imagine we receive a measurement result of 427. This is a "good" result, as the closest integer to* $5 \times (512/6) = 426\frac{2}{3}$. *The continued fraction expansion of* $z = 427/512$ *is*

$$\frac{427}{512} = \frac{1}{1 + \frac{1}{5 + \frac{1}{42 + \frac{1}{2}}}}.$$

From this we obtain the sequence of convergents

$$1, \quad \frac{1}{1 + \frac{1}{5}} = \frac{5}{6}, \quad \frac{1}{1 + \frac{1}{5 + \frac{1}{42}}} = \frac{211}{253}.$$

Only the second of these has a denominator smaller than N *and is within* $1/(2N^2)$ *of* z. *Therefore, we have* $\ell/r = 5/6$. *We output the denominator, 6, as our guess for the period* r*… which is correct!*

5.4 Complexity analysis

How complex is the final, overall algorithm? Recall that N is n bits in length. We have seen that the QFT on $\mathbb{Z}_M$ can be implemented in time $O(\log^2 M) = O(n^2)$. To implement the modular exponentiation operation $f(x) = a^x \bmod N$ efficiently, we can use repeated squaring to produce $f(2^k)$ for any integer k in k squaring operations. Multiplying the different values $f(2^k)$ together for each k such that the k'th bit of a is nonzero produces $f(x)$. So we require $O(n)$ multiplications of n-bit integers to compute $f(x)$. Multiplying two n-bit numbers can be achieved classically using standard long multiplication in time $O(n^2)$, so we get an overall complexity of $O(n^3)$.

It turns out (though we will not show it here) that the classical processing of the measurement results based on Euclid's algorithm and the continued fractions algorithm can also be done in time $O(n^3)$. Thus the overall time complexity of the whole algorithm is $O(n^3)$, whereas the best known classical algorithm runs in time exponential in $n^{1/3}$. The asymptotic quantum complexity can in fact be improved a bit further, to $O(n^2 \operatorname{poly} \log n)$, by using more advanced multiplication and division algorithms with runtime $O(n \operatorname{poly} \log n)$. However, these algorithms only become more efficient in practice for very large values of n.

6 Phase estimation

We now discuss an important primitive used in quantum algorithms called *phase estimation*, which provides a different and unifying perspective on the quantum algorithms which you have just seen. Phase estimation is once again based on the QFT over $\mathbb{Z}_N$, where $N = 2^n$.

Imagine we are given a unitary operator U. U may either be written down as a quantum circuit, or we may be given access to a black box which allows us to apply a controlled-U^j operation for integer values of j. We are also given a state $|\psi\rangle$ which is an eigenvector of U: $U|\psi\rangle = e^{2\pi i\phi}|\psi\rangle$ for some real ϕ such that $0 \le \phi < 1$. We would like to determine ϕ to n bits of precision, for some arbitrary n.

To do so, we prepend an n qubit register to $|\psi\rangle$, initially in the state $|0\rangle$, and create the state

$$\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle|\psi\rangle$$

by applying a Hadamard gate to each qubit in the first register. We then apply the unitary operator

$$U' = \sum_{x=0}^{N-1}|x\rangle\langle x| \otimes U^x.$$

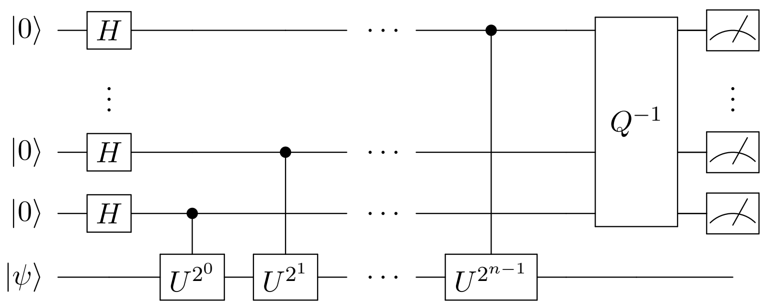
This operator can be thought of as performing the map where if the first register contains x, we apply U x times to the second register. By expressing x in binary we can implement U' using controlled-U^{2^j} gates for different integers j, controlled on different qubits in the first register. After applying U', we are left with the state

$$\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}e^{2\pi i\phi x}|x\rangle|\psi\rangle;$$

note that the second register is left unchanged. We now apply the operator Q^{-1} to the first register and then measure it, receiving outcome y (say). We output the binary fraction

$$0.y_1y_2\dots y_n = \frac{y_1}{2} + \frac{y_2}{4} + \dots + \frac{y_n}{2^n}$$

as our guess for ϕ. The following is an explicit circuit for the above algorithm.



Why does this algorithm work? When we perform the final measurement, the probability of getting an outcome x is

$$\frac{1}{N^2}\left|\sum_{y=0}^{N-1}e^{2\pi i\phi y - 2i\pi xy/N}\right|^2 = \frac{1}{N^2}\left|\sum_{y=0}^{N-1}e^{2\pi iy(\phi - x/N)}\right|^2.$$

First imagine that the binary expansion of ϕ is at most n bits long, or in other words $\phi = z/N$ for some $0 \leq z \leq N-1$. In this case we have

$$\frac{1}{N^2}\left|\sum_{y=0}^{N-1} e^{2\pi i y(\phi - x/N)}\right|^2 = \frac{1}{N^2}\left|\sum_{y=0}^{N-1} e^{2\pi i y(z-x)/N}\right|^2 = \delta_{xz}$$

by the unitarity of the QFT, so the measurement outcome is guaranteed to be z, implying that the algorithm outputs ϕ with certainty. If the binary expansion of ϕ is longer than n bits, we now show that we still get the best possible answer with probability $\Omega(1)$, and indeed are very likely to get an answer close to ϕ. The proof turns out to be very similar to that of correctness of the periodicity determination algorithm in the approximate case.

Theorem 6.1. *The probability that the above algorithm outputs the real number with n binary digits which is closest to ϕ is at least $4/\pi^2$. Further, the probability that the algorithm outputs θ such that $|\theta - \phi| \geq \epsilon$ is at most $O(1/(N\epsilon))$.*

Proof. If the binary expansion of ϕ has n binary digits or fewer, we are done by the argument above. So, assuming it does not, let $\widetilde{\phi}$ be the closest approximation to ϕ that has n binary digits, and write $\widetilde{\phi} = a/N$ for some integer $0 \leq a \leq N-1$. For any z, define $\delta(z) := \phi - z/N$ and note that $0 < |\delta(a)| \leq 1/(2N)$. For any ϕ, the probability of getting outcome z from the final measurement is

$$\Pr[z] = \frac{1}{N^2}\left|\sum_{y=0}^{N-1} e^{2\pi i y(\phi - z/N)}\right|^2 = \frac{1}{N^2}\left|\sum_{y=0}^{N-1} e^{2\pi i y\delta(z)}\right|^2 = \frac{1}{N^2}\left|\frac{1-e^{2\pi i N\delta(z)}}{1-e^{2\pi i\delta(z)}}\right|^2 = \frac{\sin^2(\pi N\delta(z))}{N^2\sin^2(\pi\delta(z))}, \tag{6}$$

where we evaluate the sum using the formula for a geometric series. This quantity should be familiar from the proof of correctness of the periodicity determination algorithm.

We first lower bound this expression for $z = a$ to prove the first part of the lemma. As $|\delta(a)| \leq 1/(2N)$, we have $N\pi\delta(a) \leq \pi/2$. Then

$$\Pr[\text{get outcome } a] = \frac{\sin^2(\pi N\delta(a))}{N^2\sin^2(\pi\delta(a))} \geq \frac{(2N\delta(a))^2}{N^2(\pi\delta(a))^2} = \frac{4}{\pi^2}$$

using the trigonometric inequalities (4).

In order to prove the second part of the theorem, we now find an *upper* bound on expression (6). First, it is clear that $\sin^2(\pi N\delta(z)) \leq 1$ always. For the denominator, by the same argument to above we have $\sin(\pi\delta(z)) \geq 2\delta(z)$ and hence, for all z,

$$\Pr[\text{get outcome } z] \leq \frac{1}{N^2}\left(\frac{1}{2\delta(z)}\right)^2 = \frac{1}{4N^2\delta(z)^2}.$$

We now sum this expression over all z such that $|\delta(z)| \geq \epsilon$. The sum is symmetric about $\delta(z) = 0$, and as z is an integer, the terms in this sum corresponding to $\delta(z) > 0$ are $\delta_0, \delta_0 + 1/N, \dots$, for some $\delta_0 \geq \epsilon$. The sum will be maximised when $\delta_0 = \epsilon$, when we obtain

$$\begin{aligned}\Pr[\text{get outcome } z \text{ with } |\delta(z)| \geq \epsilon] &\leq \frac{1}{4N^2}\sum_{k=0}^{\infty}\frac{1}{(\epsilon + k/N)^2} \leq \frac{1}{4}\int_0^{\infty}\frac{1}{(N\epsilon + k)^2}dk \\ &= \frac{1}{4}\int_{N\epsilon}^{\infty}\frac{1}{k^2}dk = O\left(\frac{1}{N\epsilon}\right).\end{aligned}$$

□

We observe the following points regarding the behaviour of this algorithm.

- What happens if we do not know an eigenvector of U? If we input an arbitrary state $|\varphi\rangle$ to the phase estimation algorithm, we can write it as a superposition $|\varphi\rangle = \sum_j \alpha_j |\psi_j\rangle$ over eigenvectors $\{|\psi_j\rangle\}$. Therefore, the algorithm will output an estimate of each corresponding eigenvalue ϕ_j with probability $|\alpha_j|^2$. This may or may not allow us to infer anything useful, depending on what we know about U in advance.

- In order to approximate ϕ to n bits of precision, we needed to apply the operator U^{2^m}, for all $0 \leq m \leq n-1$. If we are given U as a black box, this may be prohibitively expensive as we need to use the black box exponentially many times in n. However, if we have an explicit circuit for U, we may be able to find a more efficient way of computing U^{2^m}. An example of this is modular exponentiation, where we can efficiently perform repeated squaring.

6.1 Application to quantum counting

An elegant application of phase estimation is to a generalisation of the unstructured (Grover) search problem. Imagine we have an oracle $f : \{0,1\}^n \to \{0,1\}$ which takes the value 1 on k inputs, for some unknown k, and again set $N = 2^n$. We would like to estimate k by querying f.

Classically, a natural way to do this is by sampling. Imagine that we query f on q random inputs and get that f is 1 on ℓ of those inputs. Then as our estimate of k we output $\widetilde{k} = \ell N/q$. One can show using properties of the binomial distribution that this achieves

$$|\widetilde{k} - k| = O\left(\sqrt{\frac{k(N-k)}{q}}\right)$$

with high probability. We can achieve improved accuracy by using the phase estimation algorithm. Consider the "Grover iteration" $G = -H^{\otimes n} U_0 H^{\otimes n} U_f$. As G is a rotation through angle 2θ in a 2-dimensional plane, where θ satisfies $\sin\theta = \sqrt{k/N}$, its eigenvalues are $e^{2i\theta}$ and $e^{-2i\theta}$. In order to estimate k, we can apply the phase estimation algorithm to G to estimate either one of these eigenvalues. As it does not matter which we estimate, we can input any state within this 2-dimensional plane to the phase estimation algorithm as a claimed eigenvector of G. In particular, the state $\frac{1}{\sqrt{N}} \sum_{x \in \{0,1\}^n} |x\rangle$ will work.

By Theorem 6.1, if we apply the phase estimation algorithm to G, we can find the closest m-digit number to θ, for any m, with constant probability of success using $O(2^m)$ queries. For small θ, we have $\theta \approx \sqrt{k/N}$, so we learn $\sqrt{k/N}$ up to additive error $O(1/2^m)$ using $O(2^m)$ queries. Setting $2^m = \sqrt{N}/\delta$ for some real $\delta > 0$, we have learnt $\sqrt{k}$ up to additive error $O(\delta)$ using $O(\sqrt{N}/\delta)$ queries; or in other words have learnt k up to additive error $O(\delta\sqrt{k})$ using $O(\sqrt{N}/\delta)$ queries. In order to achieve a similar level of accuracy classically, we would need $\Omega(N/\delta^2)$ queries for small k.

Another application of phase estimation, to the order finding problem, is discussed in the Exercises.

7 Hamiltonian simulation

One of the earliest – and most important – applications of a quantum computer is likely to be the simulation of quantum mechanical systems. There are quantum systems for which no efficient classical simulation is known, but which we can simulate on a universal quantum computer. What does it mean to "simulate" a physical system? According to the OED, simulation is "the technique of imitating the behaviour of some situation or process (whether economic, military, mechanical, etc.) by means of a suitably analogous situation or apparatus". What we will take simulation to mean here is approximating the *dynamics* of a physical system. Rather than tailoring our simulator to simulate only one type of physical system (which is sometimes called *analogue* simulation), we seek a general simulation algorithm which can simulate many different types of system (sometimes called *digital* simulation).

According to quantum mechanics, physical systems are specified by *Hamiltonians.* For the purposes of this unit, a Hamiltonian H is a Hermitian operator acting on n qubits (i.e. $H = H^\dagger$), which corresponds physically to a system made up of n 2-level subsystems. The time evolution of the state $|\psi\rangle$ of a quantum system is governed by Schrödinger's equation:

$$i\hbar\frac{d}{dt}|\psi(t)\rangle = H(t)|\psi(t)\rangle,$$

where $H(t)$ is the Hamiltonian of the system (for convenience, we will henceforth absorb $\hbar$ into $H(t)$). An important special case on which we will focus is the *time-independent* setting where $H(t) = H$ is constant. In this case the solution of this equation is

$$|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle,$$

where (as usual) the exponential e^M for an operator M is defined by

$$e^M = \sum_{k\geq 0}\frac{M^k}{k!} = I + M + \frac{M^2}{2} + \frac{M^3}{6} + \dots.$$

Given a physical system specified by some Hamiltonian H, we would like to simulate the evolution of the system on an arbitrary initial state for a certain amount of time t. In other words, given H, we would like to implement a unitary operator which approximates

$$U(t) = e^{-iHt}.$$

What does it mean to approximate a unitary? The "gold standard" of approximation is approximation in the operator norm (aka spectral norm)

$$\|A\| := \max_{|\psi\rangle\neq 0}\frac{\|A|\psi\rangle\|}{\||\psi\rangle\|},$$

where $\||\psi\rangle\| = \sqrt{\langle\psi|\psi\rangle}$ is the usual Euclidean norm of $|\psi\rangle$. Note that this is indeed a norm, and in particular satisfies the triangle inequality $\|A + B\| \leq \|A\| + \|B\|$. We say that $\widetilde{U}$ approximates U to within ϵ if

$$\|\widetilde{U} - U\| \leq \epsilon.$$

This is a natural definition of approximation because it implies that, for any state $|\psi\rangle$, $\widetilde{U}|\psi\rangle$ and $U|\psi\rangle$ are only distance at most ϵ apart.

7.1 Simulation of k-local Hamiltonians

In order for our quantum simulation of a Hamiltonian H to be efficient, we need $U = e^{-iHt}$ to be approximable by a quantum circuit containing poly(n) gates. A fairly straightforward counting argument shows that not all Hamiltonians H can be simulated efficiently. However, it turns out that several important physically motivated classes can indeed be simulated. Perhaps the most important of these is *k-local* Hamiltonians.

A Hamiltonian H of n qubits is said to be k-local if it can be written as a sum

$$H = \sum_{j=1}^{m} H_j$$

for some m, where each H_j is a Hermitian matrix which acts non-trivially on at most k qubits. That is, H_j is the tensor product of a matrix H_j' on k qubits, and the identity matrix on the remaining $n-k$ qubits. For example, the Hamiltonian H on 3 qubits defined by

$$H = X \otimes I \otimes I - 2I \otimes Z \otimes Y$$

is 2-local. Many interesting physical systems are k-local for small k (say $k \leq 3$), some of which you may be familiar with. Simple examples include the two-dimensional Ising model on a $n \times n$ square lattice,

$$H = J \sum_{i,j=1}^{n} Z^{(i,j)} Z^{(i,j+1)} + Z^{(i,j)} Z^{(i+1,j)}$$

and the Heisenberg model on a line,

$$H = \sum_{i=1}^{n} J_x X^{(i)} X^{(i+1)} + J_y Y^{(i)} Y^{(i+1)} + J_z Z^{(i)} Z^{(i+1)},$$

both of which are used in the study of magnetism and many other areas of physics (in the equations above, $M^{(j)}$ denotes a single-qubit operator acting on the j'th qubit, and J, J_x, J_y, J_z are constants).

Note that, if H is k-local for $k = O(1)$, we can assume that $m \leq \binom{n}{k} = O(n^k)$, so m is polynomial in n. We first show that each of the individual H_j operators can be simulated efficiently, which is immediate from the following theorem, formalising a claim made at the start of the unit.

Theorem 7.1 (Solovay-Kitaev theorem). *Let U be a unitary operator which acts non-trivially on $k = O(1)$ qubits, and let S be an arbitrary universal set of quantum gates. Then U can be approximated in the operator norm to within ϵ using $O(\log^c(1/\epsilon))$ gates from S, for some $c < 4$.*

Proof. Sadly beyond the scope of this course. For a readable explanation, see Andrew Childs' lecture notes. □

As each e^{-iH_jt} acts non-trivially on only at most k qubits, it follows from the Solovay-Kitaev theorem that we can approximate each of these operators individually to within ϵ in time $O(\text{polylog}(1/\epsilon))$. In the special case where all of the H_j operators commute, we have

$$e^{-iHt} = e^{-i(\sum_{j=1}^{m} H_j)t} = \prod_{j=1}^{m} e^{-iH_jt}.$$

Thus a natural way to find a unitary operator approximating e^{-iHt} is to take the product of our approximations of $e^{-iH_1t}, \dots, e^{-iH_mt}$. Although each of these approximates e^{-iH_jt} to within ϵ, this does not imply that their product approximates e^{-iHt} to within ϵ. However, we now show that the approximation error only scales linearly.

Lemma 7.2. *Let (U_i), (V_i) be sequences of m unitary operators satisfying $\|U_i - V_i\| \le \epsilon$ for all $1 \le i \le m$. Then $\|U_m \dots U_1 - V_m \dots V_1\| \le m\epsilon$.*

Proof. The proof is by induction on m. The claim trivially holds for $m = 1$. Assuming that it holds for a given m, we have

$$\begin{aligned}
&\|U_{m+1}U_m \dots U_1 - V_{m+1}V_m \dots V_1\| \\
&= \|U_{m+1}U_m \dots U_1 - U_{m+1}V_m \dots V_1 + U_{m+1}V_m \dots V_1 - V_{m+1}V_m \dots V_1\| \\
&\le \|U_{m+1}U_m \dots U_1 - U_{m+1}V_m \dots V_1\| + \|U_{m+1}V_m \dots V_1 - V_{m+1}V_m \dots V_1\| \\
&= \|U_{m+1}(U_m \dots U_1 - V_m \dots V_1)\| + \|(U_{m+1} - V_{m+1})V_m \dots V_1\| \\
&= \|U_m \dots U_1 - V_m \dots V_1\| + \|U_{m+1} - V_{m+1}\| \\
&\le (m+1)\epsilon.
\end{aligned}$$

□

Thus, in order to approximate $\prod_{j=1}^m e^{-iH_jt}$ to within ϵ, it suffices to approximate each of the H_j to within ϵ/m. We formalise this as the following proposition.

Proposition 7.3. *Let H be a Hamiltonian which can be written as the sum of m commuting terms H_j, each acting non-trivially on $k = O(1)$ qubits. Then, for any t, there exists a quantum circuit which approximates the operator e^{-iHt} to within ϵ in time $O(m \operatorname{polylog}(m/\epsilon))$.*

7.2 The non-commuting case

Unfortunately, this simulation technique does *not* necessarily work for non-commuting H_j. The reason is that if A and B are non-commuting operators, it need not hold that $e^{-i(A+B)t} = e^{-iAt}e^{-iBt}$. However, we can simulate non-commuting Hamiltonians via an observation known as the Lie-Trotter product formula.

In what follows, the notation $X + O(\epsilon)$, for a matrix X, is used as shorthand for $X + E$, where E is a matrix satisfying $\|E\| \le C\epsilon$, for some universal constant C (not depending on X or ϵ).

Lemma 7.4 (Lie-Trotter product formula)**.** *Let A and B be Hermitian matrices such that $\|A\| \le \delta$ and $\|B\| \le \delta$, for some real $\delta \le 1$. Then*

$$e^{-iA}e^{-iB} = e^{-i(A+B)} + O(\delta^2).$$

Proof. From the Taylor series for e^x, for any matrix A such that $\|A\| = \delta \le 1$, we have

$$e^{-iA} = I - iA + \sum_{k=2}^{\infty} \frac{(-iA)^k}{k!} = I - iA + (-iA)^2 \sum_{k=0}^{\infty} \frac{(-iA)^k}{(k+2)!} = I - iA + O(\delta^2),$$

where the last equality follows from

$$\left\| \sum_{k=0}^{\infty} \frac{(-iA)^k}{(k+2)!} \right\| \le \sum_{k=0}^{\infty} \frac{\delta^k}{(k+2)!} \le e^{\delta} = O(1).$$

Hence

$$e^{-iA}e^{-iB} = \left(I - iA + O(\delta^2)\right)\left(I - iB + O(\delta^2)\right) = I - iA - iB + O(\delta^2) = e^{-i(A+B)} + O(\delta^2).$$

□

Applying this formula multiple times, for any Hermitian matrices $H_1, \dots, H_m$ satisfying $\|H_j\| \leq \delta \leq 1$ for all j,

$$\begin{aligned} e^{-iH_1}e^{-iH_2}\dots e^{-iH_m} &= \left(e^{-i(H_1+H_2)} + O(\delta^2)\right)e^{-iH_3}\dots e^{-iH_m} \\ &= \left(e^{-i(H_1+H_2+H_3)} + O((2\delta)^2)\right)e^{-iH_4}\dots e^{-iH_m} + O(\delta^2) \\ &= e^{-i(H_1+\dots+H_m)} + O(\delta^2) + O((2\delta)^2) + \dots + O(((m-1)\delta)^2) \\ &= e^{-i(H_1+\dots+H_m)} + O(m^3\delta^2). \end{aligned}$$

Applying this claim to the matrices $H_j t/p$ for arbitrary t and some large integer p, we have

$$\left\| e^{-iH_1t/p}e^{-iH_2t/p}\dots e^{-iH_mt/p} - e^{-i(H_1+\dots+H_m)t/p} \right\| = O\left(m^3\left(\frac{t\delta}{p}\right)^2\right).$$

So there is a universal constant C such that if $p \geq Cm^3(t\delta)^2/\epsilon$,

$$\left\| e^{-iH_1t/p}e^{-iH_2t/p}\dots e^{-iH_mt/p} - e^{-i(H_1+\dots+H_m)t/p} \right\| \leq \epsilon/p.$$

By Lemma 7.2, for any such p,

$$\left\| \left(e^{-iH_1t/p}e^{-iH_2t/p}\dots e^{-iH_mt/p}\right)^p - e^{-i(H_1+\dots+H_m)t} \right\| \leq \epsilon.$$

Given this result, we can simulate a k-local Hamiltonian simply by simulating the evolution of each term for time t/p to high enough accuracy and concatenating the individual simulations. We summarise this as the following theorem.

Theorem 7.5. *Let H be a Hamiltonian which can be written as the sum of m terms H_j, each acting non-trivially on $k = O(1)$ qubits and satisfying $\|H_j\| \leq 1$. Then, for any t, there exists a quantum circuit which approximates the operator e^{-iHt} to within ϵ in time $O(m^4t^2/\epsilon)$, up to polylogarithmic factors.*

It seems somewhat undesirable that, in order to simulate a Hamiltonian for time t, this algorithm has dependence on t which is $O(t^2)$. In fact, using more complicated simulation techniques, the overall complexity in Theorem 7.5 can be improved to time $O(mt)$, up to polylogarithmic factors.

8 Noise and the framework of quantum channels

Not all processes which occur in quantum mechanics are reversible. As a very simple example, consider the operation of throwing away the input state and replacing it with the state $|0\rangle$:

$$|\psi\rangle \mapsto |0\rangle$$

for all $|\psi\rangle$. This clearly cannot be implemented as a unitary operator. Just as mixed states generalise pure states, we can generalise unitary operations to so-called completely positive trace-preserving (CPTP) maps, also known as quantum channels. These occur throughout quantum information theory and are particularly useful for describing noisy operations and *decoherence*, the bane of quantum computers.

Recall that a mixed state ρ of n qubits, which describes a probabilistic mixture of pure states $|\psi\rangle$, is a Hermitian $2^n \times 2^n$ matrix which is positive semidefinite (all its eigenvalues are non-negative; we write $\rho \geq 0$ to denote this) and has trace 1. What axioms would we like a physically reasonable map ("superoperator") $\mathcal{E}$, which takes mixed states to mixed states, to satisfy?

1. $\mathcal{E}$ should be linear: $\mathcal{E}(p\rho + q\sigma) = p\,\mathcal{E}(\rho) + q\,\mathcal{E}(\sigma)$ for all real p, q.
2. $\mathcal{E}$ should be trace-preserving: $\operatorname{tr}\mathcal{E}(\rho) = \operatorname{tr}\rho$.
3. $\mathcal{E}$ should preserve positivity: if $\rho \geq 0$, then $\mathcal{E}(\rho) \geq 0$. But there is a further constraint: if we apply $\mathcal{E}$ to part of an entangled quantum state, the whole state should remain positive semidefinite. That is, $(\mathcal{E} \otimes \mathcal{I})(\rho) \geq 0$ for $\rho \geq 0$, where $\mathcal{I}$ is the identity map on an arbitrarily large ancilla system. This constraint is called *complete positivity.*

A completely positive, trace-preserving linear map is called a quantum channel.

8.1 Representations of quantum channels

There are a number of ways in which quantum channels can be represented. We describe two here.

We first consider the **Kraus** (aka "operator-sum") representation. Here a channel $\mathcal{E}$ with input dimension d_{in} and output dimension d_{out} is described by a sequence of $d_{\text{out}} \times d_{\text{in}}$ matrices E_k such that

$$\sum_k E_k^\dagger E_k = I.$$

The effect of $\mathcal{E}$ on a state ρ is then

$$\mathcal{E}(\rho) = \sum_k E_k \rho E_k^\dagger.$$

A trivial example of a channel in Kraus form is the identity channel $\mathcal{I}(\rho) = \rho$, which has one Kraus operator $E_1 = I$. For a product channel $\mathcal{E} = \mathcal{E}_1 \otimes \mathcal{E}_2$, where

$$\mathcal{E}_1(\rho) = \sum_k E_k^{(1)} \rho (E_k^{(1)})^\dagger, \quad \mathcal{E}_2(\sigma) = \sum_k E_k^{(2)} \sigma (E_k^{(2)})^\dagger$$

we have

$$\begin{aligned}\mathcal{E}(\rho \otimes \sigma) &= \mathcal{E}_1(\rho) \otimes \mathcal{E}_2(\sigma) = \left(\sum_k E_k^{(1)} \rho (E_k^{(1)})^\dagger\right) \otimes \left(\sum_\ell E_\ell^{(2)} \sigma (E_\ell^{(2)})^\dagger\right) \\ &= \sum_{k,\ell} (E_k^{(1)} \otimes E_\ell^{(2)})(\rho \otimes \sigma)((E_k^{(1)})^\dagger \otimes (E_\ell^{(2)})^\dagger),\end{aligned}$$

so $\mathcal{E}$ must have Kraus operators which are tensor products of each pair of those of $\mathcal{E}_1$ and $\mathcal{E}_2$.

We claim that any superoperator in Kraus form obeys our three axioms:

1. $\mathcal{E}$ is linear: for any X and Y,
$$\mathcal{E}(X+Y) = \sum_k E_k(X+Y)E_k^\dagger = \sum_k E_k X E_k^\dagger + \sum_k E_k Y E_k^\dagger = \mathcal{E}(X) + \mathcal{E}(Y).$$

2. $\mathcal{E}$ is trace-preserving:
$$\operatorname{tr}\mathcal{E}(\rho) = \operatorname{tr}\left(\sum_k E_k \rho E_k^\dagger\right) = \sum_k \operatorname{tr}\left(E_k\rho E_k^\dagger\right) = \sum_k \operatorname{tr}\left(\rho E_k^\dagger E_k\right) = \operatorname{tr}\left(\rho \sum_k E_k^\dagger E_k\right) = \operatorname{tr}\rho,$$
where in the second and fourth equalities we use linearity of trace, and in the third we use its invariance under cyclic shifts.

3. $\mathcal{E}$ is completely positive. Here it is sufficient to show that $\mathcal{E}\otimes\mathcal{I}$ maps positive operators to positive operators. Let ρ be an arbitrary positive operator on the extended system (with ancilla). Then
$$(\mathcal{E}\otimes\mathcal{I})(\rho) = \sum_k (E_k\otimes I)\rho(E_k^\dagger\otimes I).$$
For this to be positive semidefinite, it is sufficient that for any positive semidefinite ρ and any matrix M, $M\rho M^\dagger$ is positive semidefinite. This holds because we can expand ρ as a convex combination of pure states $|\psi_i\rangle$, and
$$M|\psi_i\rangle\langle\psi_i|M^\dagger = |\psi_i'\rangle\langle\psi_i'|$$
for some (unnormalised) vector $|\psi_i'\rangle$, which is positive semidefinite.

In fact, it turns out that the equivalence goes the other way too: every quantum channel can be written in Kraus form. We omit the proof (see Nielsen & Chuang, Theorem 8.1).

The Kraus representation is not unique. For example, the channels described by pairs of Kraus operators (with respect to the standard basis)
$$E_1 = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix},\quad E_2 = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix},\quad \text{and}\quad F_1 = \begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix},\quad F_2 = \begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix}$$
have different representations, but are actually the same channel. This can be seen by writing $F_1 = (E_1+E_2)/\sqrt{2}$, $F_2 = (E_1-E_2)/\sqrt{2}$, and calculating
$$F_1\rho F_1^\dagger + F_2\rho F_2^\dagger = \frac{(E_1+E_2)\rho(E_1^\dagger+E_2^\dagger) + (E_1-E_2)\rho(E_1^\dagger-E_2^\dagger)}{2} = E_1\rho E_1^\dagger + E_2\rho E_2^\dagger.$$

Second, we describe the **Stinespring** representation. In this representation, a channel $\mathcal{E}$ from system A to system B, $\mathcal{E}:\mathcal{B}(\mathbb{C}^{d_A})\to\mathcal{B}(\mathbb{C}^{d_B})$, is described by an isometry from A to the pair of systems B and E, $V:\mathbb{C}^{d_A}\to\mathbb{C}^{d_B}\otimes\mathbb{C}^{d_E}$, such that
$$\mathcal{E}(\rho) = \operatorname{tr}_E(V\rho V^\dagger).$$
(We say that V is an isometry if $V^\dagger V = I$; this is equivalent to V preserving inner products between states.) That is, the channel consists of applying an isometry to ρ, mapping it into a

larger-dimensional space, then tracing out (discarding) part of the space. We can think of this picture as modelling the dynamics of an open quantum system, where ρ interacts with the outside environment E, then to calculate the final state of the system we discard the environment.

Given a Kraus representation of a channel with operators E_k, we can write down a Stinespring representation in a straightforward way:

$$V = \sum_k E_k \otimes |k\rangle. \tag{7}$$

Then

$$V^\dagger V = \sum_{k,\ell} E_k^\dagger E_\ell \langle k|\ell\rangle = \sum_k E_k^\dagger E_k = I,$$

so V is indeed an isometry. Further, for any ρ,

$$\begin{aligned}\operatorname{tr}_E(V\rho V^\dagger) &= \sum_{k,\ell} \operatorname{tr}_E[(E_k \otimes |k\rangle)\rho(E_\ell^\dagger \otimes \langle \ell|)] = \sum_{k,\ell} \operatorname{tr}_E[(E_k\rho \otimes |k\rangle)(E_\ell^\dagger \otimes \langle \ell|)] \\ &= \sum_{k,\ell}(E_k\rho E_\ell^\dagger)\langle \ell|k\rangle = \sum_k E_k\rho E_k^\dagger,\end{aligned}$$

so this faithfully represents the original channel. Also observe that any isometry V can be decomposed as in Eqn. (7), so the equivalence goes both ways. As we can think of any isometry as a unitary operator on a larger space, this correspondence implies that any quantum channel can be thought of as unitary evolution of a larger system, followed by tracing out a subsystem.

8.2 Examples of quantum channels

- **Unitary evolution** is a quantum channel described by one Kraus operator: $\rho \mapsto U\rho U^\dagger$.
- Discarding ρ and **replacing it** with some state $|\psi\rangle$ is a quantum channel. If ρ is d-dimensional, we have d Kraus operators $|\psi\rangle\langle k|$, $k = 1, \dots, d$. Then, for any state ρ,

$$\sum_{k=1}^d |\psi\rangle\langle k|\rho|k\rangle\langle\psi| = |\psi\rangle \left(\sum_{k=1}^d \langle k|\rho|k\rangle\right) \langle\psi| = (\operatorname{tr}\rho)|\psi\rangle\langle\psi| = |\psi\rangle\langle\psi|.$$

- The **qubit depolarising channel** replaces the input state ρ with the maximally mixed state $I/2$, with probability p; with probability $1-p$, the input state is unchanged. That is,

$$\mathcal{E}_{\mathrm{D}}(\rho) = p\,\frac{I}{2} + (1-p)\rho.$$

  This represents a simple form of noise: with probability p, our knowledge of the qubit is completely randomised, and with probability $1-p$ the qubit is unharmed. The qubit depolarising channel has Kraus operators

$$E_1 = \sqrt{1-3p/4}I, \quad E_2 = (\sqrt{p}/2)X, \quad E_3 = (\sqrt{p}/2)Y, \quad E_4 = (\sqrt{p}/2)Z.$$

  This can be generalised to d dimensions, where we have

$$\mathcal{E}_{\mathrm{D}}(\rho) = p\,\frac{I}{d} + (1-p)\rho.$$

- The **amplitude damping channel** models a scenario in quantum optics where a photon may be lost, with some probability. We imagine that we have a basis $\{|0\rangle, |1\rangle\}$ where 0 represents "no photon" and 1 represents "photon". Then the Kraus operators of the amplitude damping channel $\mathcal{E}_{\text{AD}}$ are (with respect to the standard basis)

  $$E_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\gamma} \end{pmatrix}, \quad E_1 = \begin{pmatrix} 0 & \sqrt{\gamma} \\ 0 & 0 \end{pmatrix}$$

  for some γ, which corresponds to the probability of losing a photon. So

  $$\mathcal{E}_{\text{AD}}(|0\rangle\langle 0|) = |0\rangle\langle 0|, \quad \mathcal{E}_{\text{AD}}(|1\rangle\langle 1|) = \gamma|0\rangle\langle 0| + (1-\gamma)|1\rangle\langle 1|$$

  as expected.

- Every **measurement** is a quantum channel. Imagine we have a projective measurement, i.e. a set of orthogonal projectors $\{P_k\}$ such that $\sum_k P_k = I$. Then the probability that we get outcome k when we perform this measurement on state ρ is $\operatorname{tr} P_k\rho$, and the resulting state if this occurs is

  $$\rho'_k := \frac{P_k \rho P_k}{\operatorname{tr}(P_k\rho)}.$$

  The Kraus operators of this channel are just the projectors P_k. This is correct because

  $$\sum_k P_k\rho P_k = \sum_k \operatorname{tr}(P_k\rho)\frac{P_k\rho P_k}{\operatorname{tr}(P_k\rho)};$$

  the output state is a probabilistic mixture of the states ρ'_k with the correct probabilities.

8.3 Quantum channels and the Bloch sphere

We next discuss an intuitive geometric interpretation of quantum channels. Recall the notion of the Bloch sphere from Quantum Information Theory: that mixed states ρ of one qubit can be identified with points (called "Bloch vectors") $(\alpha_x, \alpha_y, \alpha_z)$ within the unit sphere in $\mathbb{R}^3$ via the correspondence

$$\rho = \frac{1}{2}\left(I + \alpha_x X + \alpha_y Y + \alpha_z Z\right).$$

Then it turns out that any quantum channel $\mathcal{E}$ mapping a qubit to a qubit can be thought of as an affine map on the interior of the Bloch sphere. That is, if we write $v = (\alpha_x, \alpha_y, \alpha_z)^T$, then any single-qubit channel can be associated with the map

$$v \mapsto Av + b$$

for some 3×3 matrix A and 3-dimensional vector b. To see this, first observe that as quantum channels are linear and trace-preserving, and all of the Pauli matrices have trace 0, any operator of the form $\alpha_x X + \alpha_y Y + \alpha_z Z$ must be mapped by $\mathcal{E}$ to another operator of the form $\alpha'_x X + \alpha'_y Y + \alpha'_z Z$. Applying this to a basis of 3 vectors $(\alpha_x, \alpha_y, \alpha_z)$ for $\mathbb{R}^3$ and using linearity uniquely determines A. Then, as A maps the vector $(0,0,0)$ to $(0,0,0)$, b must be the vector of coefficients corresponding to the state $\mathcal{E}(I/2)$.

So, to find A and b, we can apply $\mathcal{E}$ to $I/2$ and the Pauli matrices, and write down the Bloch vectors corresponding to the result. For example, consider the depolarising channel $\mathcal{E}_D$. Then $\mathcal{E}_{\text{D}}(I/2) = I/2$, so $b = (0,0,0)^T$. On the other hand,

$$\mathcal{E}_{\text{D}}(X) = (1-p)X, \quad \mathcal{E}_{\text{D}}(Y) = (1-p)Y, \quad \mathcal{E}_{\text{D}}(Z) = (1-p)Z,$$

so A maps the vector $(1, 0, 0)$ to the vector $(1 - p, 0, 0)$, and similarly for $(0, 1, 0)$ and $(0, 0, 1)$. Writing A as a matrix with respect to the standard basis,

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

Thus we see that, geometrically, A shrinks the Bloch sphere by a factor of $1 - p$ in every direction.

9 Quantum error-correction

Modern computer hardware is extremely reliable. Indeed, it can usually be assumed to be error-free for all intents and purposes[1]. However, early quantum computing hardware is likely to be far from reliable. Even worse, efficient quantum algorithms rely on delicate quantum effects (superposition and entanglement) which must be preserved in the presence of errors. Luckily, it turns out that errors can be fought using the notion of quantum error-correcting codes. To understand these codes, it is helpful to first consider a basic classical notion of error correction.

Imagine we have a single bit x which we would like to store in a noisy storage device. A natural model for this noise is that each bit stored in the device gets flipped with probability p, for some $0 \le p \le 1$, and is left the same with probability $1-p$. So if we store x in the device and then read it back later, we get the wrong answer with probability p. One way to improve this works as follows. Instead of just storing x, store the string xxx, i.e. repeat x three times. Then read out each of the bits of the (potentially corrupted) string to get $y := y_1y_2y_3$, and output 0 if the majority of the bits of y are 0, and 1 otherwise.

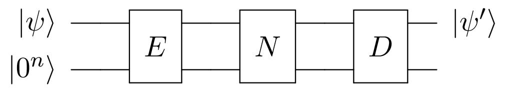
What is the probability of failing to output x if this strategy is followed? The wrong answer will be returned if two or more of the bits of y are flipped by noise, which will occur with probability $3p^2(1-p)+p^3 = p^2(3-2p) = O(p^2)$. Thus, if p is small, this strategy has improved our resistance to noise. Indeed, for any p such that $0 < p < 1/2$, we have

$$p^2(3-2p) < p,$$

so the probability of error has been reduced. Another way of looking at this situation is that we have stored a bit in such a way that it is impervious to an error affecting a single bit in the storage device. The map $x \mapsto xxx$ is a very simple example of an *error correcting code* known as the binary repetition code of length 3.

9.1 Quantum errors and error-correction

We would like to find a quantum analogue of this notion of error correction. Rather than preserving classical bits x, our quantum error correcting code should preserve a qubit $|\psi\rangle$ under some notion of error. For the time being, we pretend that an error affecting one or more qubits is simply an arbitrary and unknown unitary operator N applied to those qubits. The classical bit-flip error discussed above is an example of this, as it can be seen as simply applying the operator X to a qubit in a computational basis state (recall that $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$). The process of correcting errors in a qubit state $|\psi\rangle$ can be written diagrammatically as



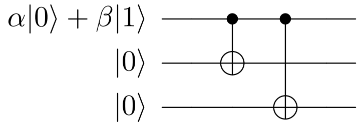
for some unitary encoding operation E, noise operation N, and decoding operation D. In other words, we encode some qubit state $|\psi\rangle$ as a larger state $|E(\psi)\rangle$ using some ancilla qubits (initially in the state $|0^n\rangle$), some noise is applied, and later we decode the noisy encoded state to produce a state $|\psi'\rangle$. The goal is that after this process $|\psi'\rangle \approx |\psi\rangle$ for some set of correctable noise operations N.

There are two obvious ways in which the classical repetition code could be translated to the quantum regime, both of which unfortunately do not work. First, we could measure $|\psi\rangle$ in the

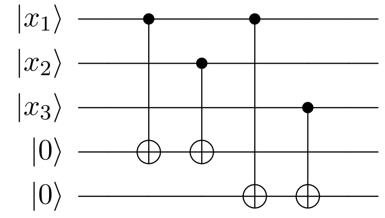
[1]Software, of course, is another matter.

computational basis to obtain a bit 0 or 1, then just encode this with the classical repetition code. This is not suitable for quantum error correction because it does not preserve quantum coherence: if $|\psi\rangle$ is in a superposition of 0 and 1 and will be used as input to a subsequent quantum algorithm, it is necessary to preserve this superposition to see any interesting quantum effects. A second idea is that we could map $|\psi\rangle \mapsto |\psi\rangle|\psi\rangle|\psi\rangle$, by analogy with the classical code. However, this is impossible (for general $|\psi\rangle$) by the no-cloning theorem.

We therefore have to take a different approach, which will be split into two steps. In the first step, we try encoding $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ as $|E(\psi)\rangle = \alpha|000\rangle + \beta|111\rangle$. Note that this is *not* the same as the "cloning" map discussed previously. Indeed, the map $\alpha|0\rangle + \beta|1\rangle \mapsto \alpha|000\rangle + \beta|111\rangle$ can be implemented using ancillas via the following simple quantum circuit.



Our decoding algorithm for this code will be based on the following quantum circuit.



Call the first three qubits the input qubits and the last two the output qubits. Following this circuit, for any basis state input $|x_1x_2x_3\rangle$, the first of the two output qubits contains $x_1 \oplus x_2$, and the second contains $x_1 \oplus x_3$. Each of these quantities is invariant under the operation of flipping all the bits of x. Thus, for any input superposition of the form $\alpha|x_1x_2x_3\rangle + \beta|x_1x_2x_3 \oplus 111\rangle$, the circuit performs the map

$$(\alpha|x_1x_2x_3\rangle + \beta|x_1x_2x_3 \oplus 111\rangle)|0\rangle|0\rangle \mapsto (\alpha|x_1x_2x_3\rangle + \beta|x_1x_2x_3 \oplus 111\rangle)|x_1 \oplus x_2\rangle|x_1 \oplus x_3\rangle.$$

This implies that, if we measure the two output qubits, we learn both $x_1 \oplus x_2$ and $x_1 \oplus x_3$ without disturbing the input quantum state. Now observe that the encoded state of $|\psi\rangle$ is always of this form, even after arbitrary bit-flip errors are applied to $|E(\psi)\rangle$:

$$\begin{aligned} |E(\psi)\rangle &= \alpha|000\rangle + \beta|111\rangle, \\ (X \otimes I \otimes I)|E(\psi)\rangle &= \alpha|100\rangle + \beta|011\rangle, \\ (X \otimes X \otimes X)|E(\psi)\rangle &= \alpha|111\rangle + \beta|000\rangle, \text{ etc.} \end{aligned}$$

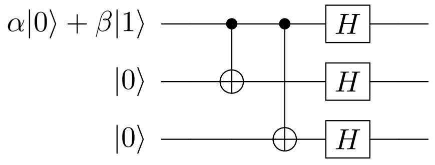
The result of measuring the output qubits is known as the *syndrome*. We now consider the different syndromes we get when different noise operators N are applied to $|E(\psi)\rangle$. First, if $N = I$ (so there has been no error applied to $|E(\psi)\rangle$), we always measure 00. On the other hand, if $N = X \otimes I \otimes I$ (i.e. a bit-flip error on the first qubit) we obtain 11 with certainty. We can write all the possible outcomes in a table as follows.

N	Syndrome
$I \otimes I \otimes I$	00
$I \otimes I \otimes X$	01
$I \otimes X \otimes I$	10
$I \otimes X \otimes X$	11
$X \otimes I \otimes I$	11
$X \otimes I \otimes X$	10
$X \otimes X \otimes I$	01
$X \otimes X \otimes X$	00

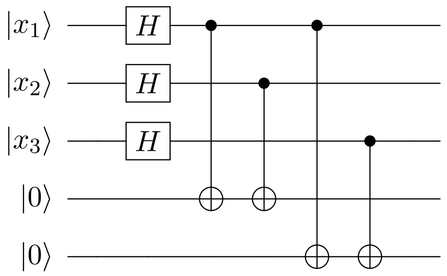
Observe that the syndromes corresponding to no error, and to bit flips on single qubits (i.e. $I \otimes I \otimes I$, $I \otimes I \otimes X$, $I \otimes X \otimes I$ and $X \otimes I \otimes I$) are all distinct. This means that, if one of these four errors occurs, we can detect it. After we detect a bit-flip error on a given qubit, we can simply apply the same bit-flip operation to that qubit to restore the original encoded state $\alpha|000\rangle + \beta|111\rangle$, which can easily then be mapped to $\alpha|0\rangle + \beta|1\rangle$ by reversing the original encoding circuit. On the other hand, if bit-flip errors occur on more than one qubit, we do not detect them (and indeed this "error correction" process can make matters worse!).

While this code is sufficient to protect against single bit-flip errors, there are other, less classical, errors acting on single qubits which it does not protect against. For example, consider the effect of a Z ("phase") error acting on the first qubit of the encoded state $|E(\psi)\rangle$, which maps it to $\alpha|000\rangle - \beta|111\rangle$ (recall that $Z = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$). It is easy to see that the syndrome measurement still returns 00, so the error correction operation does nothing and the Z error is not corrected.

However, these Z errors can be detected using a different code. Observe that $Z = HXH$, where H is the Hadamard gate. Thus Z acts in the same way as X, up to a change of basis. If we use the same code as before, but perform this change of basis for each qubit, we obtain a code which corrects against Z errors. In other words, we now encode $|\psi\rangle$ as $\alpha|+++\rangle + \beta|---\rangle$. Our new encoding circuit is simply



and our decoding circuit is



The analysis for the previous code goes through without change to show that this code protects against Z errors on an individual qubit. However, it is easy to see that the new code no longer protects against X errors! Can we protect against both errors simultaneously? The answer is yes, by *concatenating* these two codes. We first encode $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ using the code protecting against phase flips, and then encode each of the resulting qubits using the code that protects against bit flips. In other words, we perform the map

$$\begin{aligned}\alpha|0\rangle + \beta|1\rangle &\mapsto \frac{1}{2\sqrt{2}}(\alpha(|0\rangle + |1\rangle)(|0\rangle + |1\rangle)(|0\rangle + |1\rangle) + \beta(|0\rangle - |1\rangle)(|0\rangle - |1\rangle)(|0\rangle - |1\rangle)) \\ &\mapsto \frac{1}{2\sqrt{2}}(\alpha(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle) \\ &\qquad + \beta(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)).\end{aligned}$$

The single qubit $|\psi\rangle$ is now encoded using 9 qubits. These qubits can naturally be split into three blocks, each of which encodes one qubit of the state $\alpha|+++\rangle + \beta|---\rangle$. To decode this encoded state, first the decoding circuit for the bit-flip code is applied to each block. Assuming at most one bit-flip error has occurred in each block, the result will be the state $\alpha|+++\rangle + \beta|---\rangle$, perhaps with a Z error applied to one of the qubits. This state can then be mapped back to $\alpha|0\rangle + \beta|1\rangle$ using the decoding algorithm for the phase-flip code.

This quantum error-correcting code was the first such code discovered. It was invented by Peter Shor in 1995, and is (unimaginatively) known as Shor's 9 qubit code.

Example. Imagine an XZ error occurs on the fourth qubit of the encoded state (i.e. a Z error followed by an X error). The input to the decoding algorithm is thus the state

$$\frac{1}{2\sqrt{2}}(\alpha(|000\rangle + |111\rangle)(|100\rangle - |011\rangle)(|000\rangle + |111\rangle) + \beta(|000\rangle - |111\rangle)(|100\rangle + |011\rangle)(|000\rangle - |111\rangle)).$$

We apply the bit-flip decoding algorithm to each of the three blocks of three qubits, and get syndromes of 00, 11, 00 ("no error", "error on first qubit", "no error"). So we perform an X operation on the fourth qubit to correct this, and then the map $|000\rangle \mapsto |0\rangle$, $|111\rangle \mapsto |1\rangle$ on each block of three qubits. The result is the state

$$\alpha|+-+\rangle + \beta|-+-\rangle.$$

Applying the phase-flip decoding algorithm to this state gives $\alpha|0\rangle + \beta|1\rangle$ as required.

We now have a code that can protect against X or Z errors acting on an arbitrary qubit. It may seem that this is only the beginning of the gargantuan task of protecting against every one of the infinitely many possible errors that can occur. In fact, it turns out that we have already done this! The key observation is that the matrices $\{I, X, Y, Z\}$, where $Y = iXZ$, form a basis for the complex vector space of all 2×2 matrices, so an arbitrary error operation acting on a single qubit can be written as a linear combination of these matrices.

To be more precise, if we have a quantum channel $\mathcal{N}$ representing some noise process, defined by

$$\mathcal{N}(\rho) = \sum_k N_k \rho N_k^\dagger,$$

we observe that to protect $|\psi\rangle$ against $\mathcal{N}$ it is sufficient to produce an encoded state $|E(\psi)\rangle$ such that, by applying the unitary decoding operation D, $N_k|E(\psi)\rangle$ is mapped to a vector proportional to $|\psi\rangle$, for all k.

For example, imagine $\mathcal{N}$ represents noise acting on at most one qubit. Then we can expand the nontrivial part of each N_k as $N_k = \alpha_k I + \beta_k X + \gamma_k Y + \delta_k Z$. If our code protects against (say) an X error on the first qubit, we know that $D(X \otimes I \otimes \cdots \otimes I)|E(\psi)\rangle = e^{i\theta_X}|\psi\rangle$ for some θ_X. The same applies to all other single-qubit errors, with corresponding angles θ_Y, θ_Z. Consider the result of applying one Kraus operator N_k to $|E(\psi)\rangle\langle E(\psi)|$, i.e. the operator $N_k|E(\psi)\rangle\langle E(\psi)|N_k^\dagger$ (which is not in general a normalised state). We have

$$N_k|E(\psi)\rangle\langle E(\psi)|N_k^\dagger = ((\alpha_k I + \beta_k X + \gamma_k Y + \delta_k Z) \otimes I)|E(\psi)\rangle\langle E(\psi)|((\alpha_k I + \beta_k X + \gamma_k Y + \delta_k Z) \otimes I).$$

Then the result of applying the decoding operation is $DN_k|E(\psi)\rangle\langle E(\psi)|N_k^\dagger D^\dagger$. Expanding this expression, each term of the form (say) $\beta_k\delta_k(X \otimes I)|E(\psi)\rangle\langle E(\psi)|(Z \otimes I)$ must be mapped by D to something proportional to $|\psi\rangle\langle\psi|$. By linearity, the same holds for the whole operator $N_k|E(\psi)\rangle\langle E(\psi)|N_k^\dagger$; and summing over k, the same holds for the state $\mathcal{N}(|E(\psi)\rangle\langle E(\psi)|)$. This implies that $\mathcal{N}(|E(\psi)\rangle\langle E(\psi)|)$ is decoded to $|\psi\rangle\langle\psi|$, so our code in fact can correct an arbitrary error on an individual qubit.

The following general statement about when quantum error-correction is possible is known, but we will not prove it here.

Theorem 9.1 (Quantum error correction criterion)**.** *Assume we have a code subspace with basis $\{|\psi_i\rangle\}$. A necessary and sufficient condition for the set of errors $\{E_a\}$ to be correctable is*

$$\langle\psi_i|E_a^\dagger E_b|\psi_j\rangle = C_{ab}\delta_{ij}$$

for all a, b, i and j.