This handout follows a combination of Preskill’s text and the Rieffel article (see the course web page for links). Some of the text has been paraphrased from the Reiffel article.

1 Linear operators and matrices

We’ve been writing quantum logic gates as boxes with unitary transformations written inside of them. But unitary transformations are linear operators, and linear operators can all be represented as matrices. For example, the unitary operator $H$ (associated with the “Hadamard gate”) is defined as follows:

\[
H(|0\rangle) = \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle, \\
H(|1\rangle) = \frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle.
\]

But if we write our input $a|0\rangle + b|1\rangle$ as the vector \[
\begin{pmatrix} a \\ b \end{pmatrix},
\]
then we can associate with $H$ the matrix

\[
\begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix}.
\]
where one operates on a column vector with that matrix by multiplying the matrix by the column vector.

2 Simulating classical logic gates

A classical logic gate computes a function from \( n \) inputs to \( m \) outputs. We’ll focus on the case \( m = 1 \). In that case, a classical logic gate computes a function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \). Notice that no quantum circuit computes such a function directly for \( n > 1 \), for the trivial reason that quantum logic gates all have the same number of inputs as outputs.

But there’s a trick for “packaging” (Preskill’s term, whose discussion, by the way, I follow here) such a function within a unitary transformation. The trick is to construct a unitary transformation \( U_f \) that has an extra input line that starts out as \( |0\rangle \), and during the computation gets filled in with \( |0\rangle \) or \( |1\rangle \) according to whether \( f \) takes the value 0 or 1.

In other words, we set things up so that for any \( n \)-bit input \( x_1 \otimes \cdots \otimes x_n \),

\[
U_f(x_1 \otimes \cdots \otimes x_n \otimes |0\rangle) = U_f(x_1 \otimes \cdots \otimes x_n \otimes |f(0)\rangle).
\]

We can abbreviate the above as

\[
U_f(x \otimes |0\rangle) = U_f(x \otimes |f(0)\rangle).
\]

3-bit controlled-not gates, cunningly arranged, are sufficient to construct such a \( U_f \) for any \( f \). (To get the basic idea behind how to do this, think about how one might simulate AND, OR, and NOT gates using appropriately arranged controlled-not gates. Of course one may need special “constant 1” input lines, and one may need to ignore output “garbage” lines in order to do so.)
3 Core idea of Grover’s database search algorithm

(Exposition follows the Rieffel article.)

The problem: we’ve have a “black box” that computes some unknown function $f : \{0, 1\}^n \rightarrow \{0, 1\}$. We want to find an $x$ such that $f(x) = 1$ (or if there isn’t one, we want to know that). Think of the n-digit binary strings as database entries. Think of $f$ as answering the question “this is the database item we’re looking for?”

Let’s suppose that $f$ is slow—it takes ten years to spit out an answer. So we’d like to find our answer by activating $f$ as few times as possible.

Classically, we’d have to query every single entry. (Think: Time(N), where $N = 2^n$ is the number of database entries.) But quantum-ly, we can get the answer with high probability by activating $U_f$ on the order of $\sqrt{N}$ times.

First idea (which doesn’t quite work): attach a Hadamard gate to the first n inputs, and feed in n+1 $|0\rangle$’s. What is the result? Why is that not good enough.

Second idea: Do the above, but then do an operation that we can describe relative to the standard basis as follows. The operation has the effect of decreasing the absolute value of coefficient of the terms of the form

$$x_1 \otimes \cdots \otimes x_n \otimes |0\rangle$$

and increasing the absolute value of coefficient of the terms of the form

$$x_1 \otimes \cdots \otimes x_n \otimes |1\rangle$$

In other words, the terms associated with “hits”—database entries that we’re looking for—get their coefficients jacked up. Then when we measure with respect to the ordinary basis, we
get a high probability of a hit (if there is at least one).

The amplification operation consists of repeatedly doing the following two things:

1. Flipping the signs of the terms associated with “hits”.
   (This involves an additional application of $U_f$.)

2. Performing the \textit{inversion about the average} operation, which has the effect of moving each term farther away from the average value of all of the terms.

We only need to repeat the amplification operation on the order of $\sqrt{N}$ times in order to get the probability of failure to be low. For example (paraphrasing Reiffel p. 30), if there is exactly one database entry that we’re searching for, then the chance of failure is $2^{-n}$. 