Build your own quantum computers, Part 1: 3-qubit projects and Grover's quantum search [1]

D. Candela

August 11, 2016

[1] These handouts were created for the honors colloquium for Physics 424. Most of the material in Parts 1 and 2 is in the published article: American Journal of Physics **83**, 688 (2015).

"Frans had not only taken his old AI programme a step further, he had also developed fresh algorithms and new topographical material for quantum computers."

"I'm not sure I follow."

"Quantum computers are computers based on quantum mechanics. They are many thousand times faster in certain areas than conventional computers. The great advantage with quantum computers is that the fundamental constituent quantum bits—qubits—can superposition themselves."

"They can *what*?"

"Not only can they take the binary positions one or zero as do traditional computers, they can also be both zero and one at the same time. For the time being quantum computers are much too specialized and cumbersome. But—how can I best explain this to you?—Frans appeared to have found ways to make them easier, more flexible and self-learning. He was onto something great, at least potentially. But as well as feeling proud of his breakthrough, he was also worried—and that was the reason he called Steven Warburton."

"Why was he worried?"

"In the long term, because he suspected his creation could become a threat to the world, I imagine. But more immediately because he knew things about the NSA." [2]

[2] From *The Girl in the Spider's Web* by David Lagercrantz (2015).

Introduction

You may have heard the assertion that a functional quantum computer would be fantastically powerful, able to defeat current methods for data encryption. Now that you have a working knowledge of quantum mechanics, you are in a good position to find out exactly what a quantum computer is, how it works, and to form your own opinion about its future promise.

In Part 3 of these handouts you can see that one of the most promising quantum computing architectures is topological, and also that algorithms have recently been developed for quantum AI (deep learning). So perhaps the quote above from a work of fiction is not so fanciful.

In the projects described here, you will write computer programs to simulate a quantum system capable of carrying out quantum computations – you will build your own quantum computers. It will turn out that the quantum system you need to simulate is a collection of N spin-1/2 particles, so you already are familiar with the necessary physics. What may not yet be clear is how such a physical system can act as a powerful computer.

In some years the P424 honors colloquium started with simulation of a quantum mechanical particle in one dimension before moving on to quantum computing as described in these handouts. If not, a brief warm-up project will be assigned so that everyone is ready to code.

It will probably not surprise you to learn that a simulation of a quantum computer that runs on a classical computer (as you will create) is no more powerful than the classical computer. Real quantum computers have been constructed, but to date these have been

The terms "real quantum computer" or "physical quantum computer" will be used interchangeably to mean physical systems that operate quantum mechanically to carry out computations in the same manner as your simulated quantum systems.

small quantum computers with even less computational power than your simulations will have. This may not be true for much longer, however.

Necessary reading

To get started, you should read both of the following:

- The article by Gibney[3] which is a short, readable summary of progress (as of late 2014) in building real quantum computers.
- The Wikipedia article *Quantum computing*. It is not necessary at this stage to understand everything in this article in detail.

[3] Elizabeth Gibney. Quantum computer quest. *Nature*, 516:24–26, 2014

Programming languages

You can carry out these quantum computing simulations using your programming platform of choice; Python (with NumPy), MATLAB, C++, and Mathematica are all good choices.

I used Mathematica to check that all of the projects work, but I do not recommend learning this rather odd and cranky language just to do these projects.

If powerful quantum computers ever become available, you will want to use a *quantum computing language* to program it efficiently. Several such languages have already been devised, and you may be interested in reading about them or trying them out[4]—but that is not the goal of the projects described here. Similarly, a number of software packages and even graphical user interfaces have been published to simulate quantum computers.[5] In the projects here, you will build your own quantum computer simulator rather than using one of these published simulators, thereby getting a deeper understanding of quantum computing (I hope).

[4] Brian Hayes. Programming your quantum computer. *American Scientist*, 102:22–25, 2014

[5] Quantiki article *List of QC simulators*, `http://www.quantiki.org/wiki/List_of_QC_simulators`

Quantum computing paradigm

The paradigm for quantum computing explored in these projects is the *quantum gate array* or *quantum circuit* (Fig. 1). The quantum gate array can be used to carry out Grover's search algorithm, Shor's factoring algorithm, and other algorithms such as the solution of systems of linear equations. There are a number of other promising paradigms for quantum computing that are not covered Parts 1 and 2 of these handouts (adiabatic quantum computing, quantum annealing, quantum emulation. . .[6]). The important (and tricky) topic of quantum error correction is addressed in Part 3.

[6] Wikipedia article *Quantum computer*

Straightforward and more challenging projects

Many interesting projects can be carried out with $N = 3$ qubits, and the projects in Part 1 all use $N = 3$. The 8-element state vectors and 8×8 matrices can be written out explicitly, leaving some of the thornier computational issues to the later projects. Using $N = 3$ you can explore one-qubit gates like the Hadamard gate and the phase-shift gate and two-qubit gates like the CNOT. Also, Grover's quantum search can be fully implemented with $N = 3$. If you complete the $N = 3$ projects you should learn quite a lot about quantum computing, multiparticle quantum states, and quantum measurement.

Part 2 of these handouts covers projects with $N > 3$, which are more challenging. As N increases the matrices become too large to enter manually and must be computer-generated. In this way an inexpensive computer can handle up to about $N = 12$. If the additional step is taken of storing the matrices in sparse form, it is possible to reach $N \approx 20$.

Part 2 also covers Shor's quantum factoring algorithm. Shor's algorithm is more complicated to program than Grover's algorithm. To help you over this hurdle, the smallest possible Shor's-algorithm computation (using $N = 7$ to factor 15) is described in great detail. Finally, information is provided so you can try Shor's algorithm with as many qubits as your hardware can handle.

Part 3 of these handouts covers quantum error correction, which will be essential for any real quantum computer. This is a fascinating but complex and evolving subject, and you will need to add some new ingredients to your code to tackle the projects in Part 3.

Computer projects, programming hints, and exercises

There are eight programming projects in Parts 1 and 2 of these handouts, each preceded by explanatory material. The projects build on each other, with the exception of Project 6 (sparse matrices) which is useful but not required for the later projects. The Appendix has programming hints based on questions that have tended to come up in the past. A few exercises are also given. They are not difficult but they require key conceptual points to be absorbed, so you should do the exercises and include the answers in the material you turn in.

Part 3 continues with nine further programming projects all on quantum error correction, ranging from easy to quite difficult.

Sources for further information

A comprehensive source on all aspects of quantum information is the classic text by Nielsen and Chuang.[7] A number of shorter texts intended for science and engineering students have appeared; two that appear readable are Le Bellac[8] and Mermin.[9] Among the most convenient resources are the Wikipedia articles on quantum computing, Grover's algorithm and Shor's algorithm.

[7] Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, Cambridge, 2000

[8] Michel Le Bellac. *A Short Introduction to Quantum Information and Quantum Computation*. Cambridge University Press, Cambridge, 2006

[9] N. David Mermin. *Quantum Computer Science: An Introduction*. Cambridge University Press, Cambridge, 2007

The N-qubit register

The *bit* is the smallest unit of classical information, with two possible values (0 or 1). In a classical computer, a physical system like a tiny capacitor is used to store each bit, with different values of the charge on the capacitor used to represent 0 and 1. For a quantum computer, the bit is replaced by the *qubit*, stored by a quantum system with two basis vectors in its Hilbert space. An example would be a spin-1/2 particle like an electron, when only the spin can change. The general quantum state of a spin-1/2 system is

The particle could have a spatial wave function $\Psi(x)$ that stays fixed, for example.

$$|\Psi\rangle = a|\uparrow\rangle + b|\downarrow\rangle \tag{1}$$

where $|\uparrow\rangle, |\downarrow\rangle$ are the states with $S_z = \pm\hbar/2$. The complex amplitudes a and b obey the normalization condition $|a|^2 + |b|^2 = 1$. The S_z states are used to represent 0 and 1,

$$|0\rangle = |\uparrow\rangle, \quad |1\rangle = |\downarrow\rangle. \tag{2}$$

The significance of a superposition state like $\frac{1}{\sqrt{2}}(|\uparrow\rangle + |\downarrow\rangle)$ or $\frac{1}{\sqrt{2}}(|\uparrow\rangle + i|\downarrow\rangle)$ is less obvious.

An N-qubit register is N of these 2-state systems, considered together to be one quantum system. Quantum mechanics tells us that a system of N particles should be described by a single joint quantum state $|\Psi\rangle$. For example a 3-qubit register has $2^N = 8$ basis states,

$$\begin{aligned} |000\rangle &= |\uparrow\rangle|\uparrow\rangle|\uparrow\rangle \\ |001\rangle &= |\uparrow\rangle|\uparrow\rangle|\downarrow\rangle \\ |010\rangle &= |\uparrow\rangle|\downarrow\rangle|\uparrow\rangle \\ &\vdots \\ |111\rangle &= |\downarrow\rangle|\downarrow\rangle|\downarrow\rangle. \end{aligned} \tag{3}$$

Important: The *order* of the basis states is as in binary counting, $000, 001, 010, \ldots 111$ which are the 3-bit binary representations of $0, 1, 2, \ldots 7$. This would be a good time to review binary numbers and counting.

In the basis state $|001\rangle$ qubits 1 and 2 have $S_z = +\hbar/2$ while qubit 3 has $S_z = -\hbar/2$. The general quantum state of a 3-qubit register is a superposition of the eight basis states,

$$|\Psi\rangle = a|000\rangle + b|001\rangle + \cdots + g|110\rangle + h|111\rangle \tag{4}$$

where the complex amplitudes satisfy $|a|^2 + |b|^2 + \cdots + |h|^2 = 1$ for normalization. The quantum state will be stored in the computer as a column vector of eight complex numbers:

$$|\Psi\rangle \leftrightarrow \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}. \tag{5}$$

In most computer languages $|\Psi\rangle$ will simply be an eight-element array of complex numbers. In matrix multiplications $|\Psi\rangle$ must be treated as a column vector: when multiplied on the *left* by a square matrix, the result is another column vector.

The $\leftrightarrow$ symbol is used here to indicate that the same quantum state $|\Psi\rangle$ would be represented by a different vector of eight numbers if we changed to a different basis for Hilbert space. The basis used throughout this handout is the one shown in Eq. 3, which is often called "the computational basis." If you don't want to be so picky you can use $=$ in place of $\leftrightarrow$.

Exercise: Compute the largest N for which the quantum state of an N-qubit register can be stored on a 2014-era laptop computer with 4 GB = 4×10^9 bytes of RAM. Assume each complex amplitude $a, b \ldots$ requires 16 bytes of storage. How does the amount of memory needed change if the number of qubits is doubled?

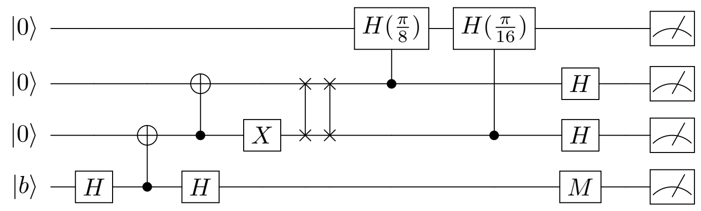


Figure 1: Example of a quantum gate array computation. Four qubits are used ($N = 4$), shown by the four horizontal lines. The diagram is read from left to right. First the qubit register is initialized to the state $|\Psi\rangle = |000b\rangle$. (Here the quantum state $|b\rangle$ is varied to provide input data to the calculation. For many other quantum calculations, the initialization state $|\Psi\rangle$ is fixed.) Next a sequence of quantum gate operations is carried out, denoted by the various symbols between the left and right sides of the diagram. Finally the qubits are measured as indicated by the voltmeter symbols on the right side. This quantum computer was built by Cai, et al. using photons as the qubits and used to solve a system of linear equations (X.-D. Cai, et al., Phys. Rev. Lett. **110**, 230501 (2013)).

The quantum gate array computer

The type of quantum computation that will be simulated in these projects is shown in Fig. 1. First the N-qubit register is initialized to a definite quantum state. Next, a series of quantum gate operations is applied to the register. Typically each gate operates on only a few (1-3) of the qubits. Finally, the qubits are measured. To simulate a quantum computer, it is necessary to carry out each of the three stages (initialization, quantum gate operations, and measurements) on a quantum state represented in the computer in the form of Eq. 5.

The initialization step is simple. Most often the N-qubit register will be initialized to one of its 2^N basis states, with all N qubits in states of definite S_z. This is represented by having one of the 2^N amplitudes $a, b, c \ldots$ equal to one, with the rest of the amplitudes equal to zero. For example the initial state might be chosen to be

$$|\Psi\rangle = |011\rangle \leftrightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{6}$$

Be sure you understand this equation before proceeding. Why is the 1 in the fourth position?

Another type of state encountered in quantum computing is the "cat state"

$$|\Psi\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle) \leftrightarrow \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1/\sqrt{2} \end{bmatrix}. \tag{7}$$

This highly-nonclassical state is named for Schrödinger's cat, as it is an equal superposition of two very different classical states.

The two-qubit cat state $(|00\rangle + |11\rangle)/\sqrt{2}$ is also called a "Bell state", as it is one of four maximally-entangled basis states, useful when discussing Bell's inequality (Townsend 2nd Ed. Eq. 5.67).

The three-qubit cat state of Eq. 7 is also called a Greenberger-Horne-Zeilinger (GHZ) state.

Measurements of the N-qubit register

Usually one measures S_z for each of the N qubits. Each measurement gives $S_z = \pm\hbar/2$. Therefore one is sure to find the register to be in one of the 2^N basis states of Eq. 3, no matter what the state $|\Psi\rangle$ was before the measurement. The *probability* of measuring each basis state is given by the square magnitude of the corresponding amplitude, for example

$$P(|110\rangle) = |\langle 110|\Psi\rangle|^2 = |g|^2. \tag{8}$$

If the result of the measurement is $|110\rangle$ then quantum mechanics tells us that $|\Psi\rangle$ *collapses* to the basis state $|110\rangle$, and subsequent measurements will all give $|110\rangle$. Therefore quantum algorithms must be cleverly devised so that a single measurement of the N-qubit register gives the needed results.

Programming project 1: Simulate measurement of the N-qubit register.

In this section, the first and last parts of the quantum gate array computer are set up: Initialization of the qubit register (left side of the diagram), and measurement of the qubits (right side of the diagram). In later sections quantum gates will be added. For the first part of this article the number of qubits is $N = 3$, so there will be three horizontal lines in the diagrams for computations. Here is what the computer program must do:

1. Allocate a column vector with $2^N = 8$ complex entries and set it to the desired initial state $|\Psi\rangle$ (Eq. 5). Some things to try are listed below.

2. Produce a result by making a simulated measurement of S_z for the three qubits. The result will be *random*, but the probability of getting each possible result must follow the quantum-mechanical law of Eq. 8. The result will always be one of the eight basis states $|000\rangle \ldots |111\rangle$, and should be printed out in this form (see Appendix for hints). For example a result of $|101\rangle$ shows that the first and third qubits were measured to be 1, while the second qubit was measured to be 0.

3. Repeat step 2 many times to see how variable the results are. It is helpful to make the program list what the most frequent results are.

If desired the program can compute $|\Psi\rangle$ once just before the measurement and then use this $|\Psi\rangle$ as the input for multiple measurements. This would not be possible in a physical quantum computer since measuring $|\Psi\rangle$ would collapse the quantum state.

Once the program is working, try the following:

- Set the initial state $|\Psi\rangle$ to one of the basis states, for example $|011\rangle$ (Eq. 6). With this initial state, every measurement should give the result $|011\rangle$.

- Set the initial state to the cat state, Eq. 7. Now, at random, either all of the qubits should be 0, or all of the qubits should be 1. Thus each qubit is equally likely to be measured 0 or 1, but all three qubits are always measured to have the same value.

- Set the initial state to an equal superposition of all 2^N basis states,

$$|\Psi\rangle \leftrightarrow \frac{1}{\sqrt{8}}\begin{bmatrix}1\\1\\1\\1\\1\\1\\1\\1\end{bmatrix}. \tag{9}$$

  This type of state will prove important later for Grover's algorithm and Shor's algorithm. With this state the measured value for each qubit is both random and uncorrelated with the other qubits. Thus, all possible results from $|000\rangle$ to $|111\rangle$ should occur, each with equal frequency to within statistical fluctuations.

Quantum gates

Classical computers are made up of logic gates with names like AND, OR, NOT. . . connected by wires that carry the states of classical bits from the output of one gate to the input of the next (Fig. 2). The quantum gate array computer (Fig. 1) is similar: quantum gates (symbols in the central part of Fig. 1) take quantum signals in from the left and produce outputs to the right. However, there are important differences:

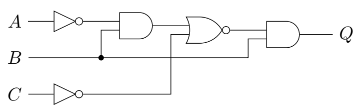


Figure 2: A classical logic circuit. The open shapes are logic gates (here two NOTs, two ANDs and a NOR) and the lines are wires carrying logic signals from one gate to another. This circuit takes boolean inputs A, B, and C and produces boolean output Q.

- Each horizontal line in Fig. 1 represents a qubit as time proceeds, not a wire in space. Therefore the quantum gates are a series of operations carried out one after another in time.
- The operations carried out by the quantum gates must *not* collapse the state of the system the way a measurement would.

A physical system that is not measured or otherwise disturbed obeys the Schrödinger equation

$$i\hbar \frac{d|\Psi\rangle}{dt} = \widehat{\mathcal{H}}|\Psi\rangle. \tag{10}$$

A symbol with a hat like $\widehat{U}$ stands for an *operator* that transforms the quantum state vector $|\Psi\rangle$ to a different state vector $|\Psi'\rangle = \widehat{U}|\Psi\rangle$.

The connection between the unitary time-evolution operator $\widehat{U}$ and the Hamiltonian operator $\widehat{\mathcal{H}}$ is discussed in detail in Townsend, *A Modern Approach to Quantum Mechanics, Second Ed.* Sec. 4.1.

In a physical quantum computer the Hamiltonian $\widehat{\mathcal{H}}$ must vary with time so that Eq. 10 results in the desired quantum gate operations. Applying the Schrödinger equation over a time period is always equivalent to applying a *unitary operator* to $|\Psi\rangle$. This gives the following important rule: **Every quantum gate operation is carried out by applying some unitary operator to the state vector,**

$$|\Psi\rangle \Leftarrow \widehat{U}|\Psi\rangle \tag{11}$$

where $\widehat{U}^\dagger\widehat{U} = \widehat{I}$ (the definition of "unitary") and $\widehat{I}$ is the identity operator.

The left-arrow notation e.g. $a \Leftarrow a+1$ is used to denote the assignment of a new value to the program variable a. In most computer languages this is written `a=a+1`.

The Hadamard gate

One of the most commonly used quantum gates is the *Hadamard gate*, symbolized a box with an H in it. There are four Hadamard gates in Fig. 1. A Hadamard gate is described by the matrix

$$\widehat{H} \leftrightarrow \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{12}$$

Again the $\leftrightarrow$ symbol is used to indicate that $\widehat{H}$ is an operator on Hilbert space, while the matrix shown is the representation of that operator in a particular basis (the computational basis).

The symbol $\widehat{H}$ is used in this paper for a Hadamard gate, not the Hamiltonian.

Exercise: Show that $\widehat{H}$ is unitary. The the matrix for the adjoint $\widehat{H}^\dagger$ is found by transposing the matrix for $\widehat{H}$ and then complex- conjugating every matrix element.

The standard basis states are

$$|0\rangle \leftrightarrow \begin{bmatrix}1\\0\end{bmatrix}, \quad |1\rangle \leftrightarrow \begin{bmatrix}0\\1\end{bmatrix}. \tag{13}$$

Applying a Hadamard gate to one basis state gives a superposition of *both* basis states,

$$\widehat{H}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle). \tag{14}$$

A Hadamard gate can also take a superposition of the basis states and "put it back together" into a single basis state, for example

$$\widehat{H}\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |1\rangle. \tag{15}$$

Exercise: Verify Eqs. 14-15 by matrix multiplication.

The phase shift gate

Another important one-bit gate is the *phase shift gate*. The matrix for this gate is

$$\widehat{R}_\theta \leftrightarrow \begin{bmatrix}1 & 0\\0 & e^{i\theta}\end{bmatrix} \tag{16}$$

with θ real. Usually θ has a specified value like $\pi/4$ or π, and the symbol for this gate is a box with θ in it. Consider the result of applying a phase-shift gate to a superposition of the basis states:

$$\text{If } |\Psi\rangle \leftrightarrow \begin{bmatrix}a\\b\end{bmatrix}, \text{ then } \widehat{R}_\theta|\Psi\rangle \leftrightarrow \begin{bmatrix}a\\e^{i\theta}b\end{bmatrix}. \tag{17}$$

Since $|e^{i\theta}b|^2 = |b|^2$, this shows that the *probabilities* for finding the qubit in the two basis states are unchanged by the phase-shift gate, but the relative *phase* of the two amplitudes is changed. Although a quantum phase cannot be measured directly, the phase shift in Eq. 17 can have an effect if further quantum gate operations are applied to the qubit before it is measured.

Applying a gate to the N-qubit register

The N-qubit register has 2^N basis states, so a quantum gate is represented by a $2^N \times 2^N$ unitary matrix. How is this larger matrix computed from the 2×2 matrix of Eq. 12 or Eq. 16? The answer is written formally as a *tensor product*. With $N = 3$ qubits the operation that applies a Hadamard gate to qubit 1 is

$$\widehat{H}^{(1)} = \widehat{H} \otimes \widehat{I} \otimes \widehat{I}. \tag{18}$$

The superscript on $\widehat{H}^{(1)}$ indicates that the gate operates on qubit 1. It is computed as the tensor product of the Hadamard operator with two copies of the identity operator. Conceptually, the identity operators "do nothing" to qubits 2 and 3. Similarly, the operators that apply a Hadamard gate to qubit 2 or qubit 3 are

$$\begin{aligned} \widehat{H}^{(2)} &= \widehat{I} \otimes \widehat{H} \otimes \widehat{I}, & (19) \\ \widehat{H}^{(3)} &= \widehat{I} \otimes \widehat{I} \otimes \widehat{H}. & (20) \end{aligned}$$

It is not necessary at this point to fully understand the tensor-product notation of Eqs. 18-19 (it is discussed further in Part 2 of these handouts). It will suffice to have the the following matrices for a Hadamard gate operating on qubit 1, 2 or 3:

$$\widehat{H}^{(1)} \leftrightarrow \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}, \tag{21}$$

$$\widehat{H}^{(2)} \leftrightarrow \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}, \tag{22}$$

$$\widehat{H}^{(3)} \leftrightarrow \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}, \tag{23}$$

Comparing these with Eq. 12, the matrix elements of $\widehat{H}$ are distributed around these larger matrices following a definite pattern. Similarly, matrices can be written for a phase-shift gate on qubit 1, 2 or 3. Here is the matrix for a phase-shift gate on qubit 3:

$$\widehat{R}_{\theta}^{(3)} \leftrightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & e^{i\theta} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{i\theta} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{i\theta} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\theta} \end{bmatrix}. \tag{24}$$

The matrix for $\widehat{R}_{\theta}^{(3)}$ follows the same pattern as $\widehat{H}^{(3)}$, with four copies of the original 2×2 matrix arrayed along the diagonal of the 8×8 matrix.

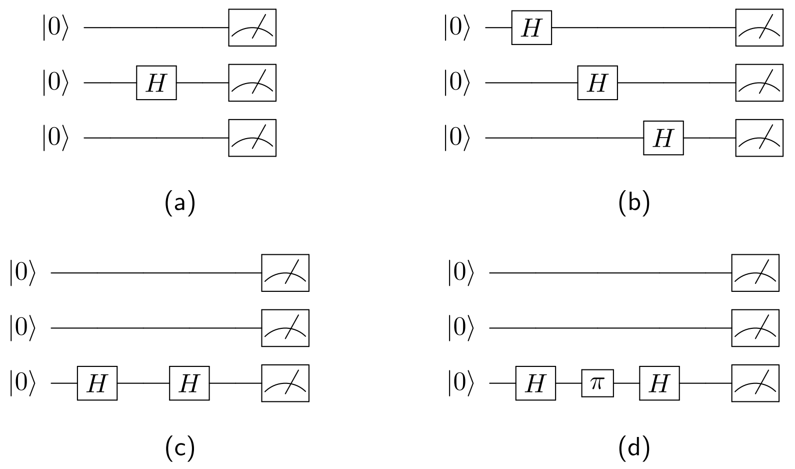


Figure 3: Four different three-qubit calculations to try.

Programming project 2: First full quantum computations.

Set up and run the four quantum computations shown in Fig. 3. **Note that all of the 8×8 matrices needed for projects 2-4 are written out explicitly in these handouts.** In Fig. 3(a) $|\Psi\rangle$ is initialized to the state $|000\rangle$, then a Hadamard gate is applied to qubit 2 by multiplying $|\Psi\rangle$ on the left by $\widehat{H}^{(2)}$:

Therefore, you do not need to compute any matrices for quantum gates, but you do need to type them into your code. Alternatively, you may want to look ahead to the the beginning of Part 2 to see how to make your code generate these matrices.

$$|\Psi\rangle \Leftarrow \widehat{H}^{(2)}|\Psi\rangle. \tag{25}$$

As noted above the $\Leftarrow$ implies that $|\Psi\rangle$ will be *replaced* by $\widehat{H}^{(2)}|\Psi\rangle$ in your computer program.

Eq. 22 gives the matrix for $\widehat{H}^{(2)}$. A more complex computation is performed by successively multiplying the quantum state on the left by operators for the specified gates. For example, the gates in Fig. 3(b) are carried out by the operation

$$|\Psi\rangle \Leftarrow \widehat{H}^{(3)}\widehat{H}^{(2)}\widehat{H}^{(1)}|\Psi\rangle. \tag{26}$$

The gates are applied to $|\Psi\rangle$ in the same order shown in the quantum circuit diagram. They appear in reverse order in Eq. 26 because each successive matrix is applied on the left. As before the computation is repeated to see what is definite and what varies randomly.

Gates can be moved along their lines as long as the order of operations *on each line* is unchanged. So for Fig. 3(b) the three Hadamard gates could be applied in any order.

- In Fig. 3(a), a Hadamard gate is applied to qubit 2. From Eq. 14 this puts qubit 2 in an equal superposition $|0\rangle$ and $|1\rangle$. Therefore the result of the calculation should vary randomly between the two possibilities $|000\rangle$ and $|010\rangle$.
- In Fig. 3(b), Hadamard gates are applied to qubits 1, 2 and 3. This leaves each of the three qubits equally likely to be in the states $|0\rangle$ and $|1\rangle$, with no correlations between the qubits. Knowing the state of qubit 1, for example, gives no information about the states of qubits 2 and 3. The result of the calculation should vary randomly between all eight possibilities $|000\rangle, |001\rangle, \ldots, |111\rangle$. Putting the N-qubit register into an equal superposition of all 2^N basis states by applying a Hadamard gate to each qubit is one of the important building blocks of many quantum algorithms.
- In Fig. 3(c), two successive Hadamard gates are applied to the same qubit. As shown above, the Hadamard gate splits a single quantum amplitude into two but it can also put two quantum amplitudes back together into one. In this case the second Hadamard undoes the effect of the first one, so the result of the calculation should always be $|000\rangle$.
- In Fig. 3(d), again two Hadamard gates are applied to the same qubit. But now the phase-shift gate $\widehat{R}_\theta$ with $\theta = \pi$ is applied between the two Hadamard gates, shown by a box with π in it. As in the previous case, the result of the calculation is perfectly definite, but now the result is always $|001\rangle$. The net effect of the three gates has been to flip qubit 3 from $|0\rangle$ to $|1\rangle$. Eqs. 14-15 show why this works. This computation shows that using a phase-shift gate to change the *phase* of quantum amplitudes can indeed change the results of the calculation.

With these programs, a complete quantum computer has been simulated. In the next section it is shown how, with a few additions, Grover's quantum search can be carried out.

Grover's quantum search algorithm

Consider a classical database like a phone book with D names, each followed by a phone number. If a phone number is given, how many entries of the phone book must be looked at to find the corresponding name? Looking through the phone book one might be lucky and find the phone number on the first try, or unlucky and need to look at all D entries. The *average* number of entries consulted will be halfway between these extremes, $D/2$.

Here is another classical search problem: A logical function of N boolean (T or F, meaning True or False) variables like Fig. 2 is specified, and it is known that the function is T only for one specific combination of the inputs (this is true for Fig. 2). To find this combination among the $D = 2^N$ possibilities for the inputs, one goes through the D possibilities until the one is found that yields T. Again, on average it is necessary to go through $D/2$ of the possibilities.

Using Grover's quantum search algorithm[10] the average number of times the database must be consulted is reduced from $D/2$ to $(\pi/4)\sqrt{D}$. This is an enormous savings if the size D of the database is large. However, there is a catch: The database or logic function must be put in the form of a *quantum oracle*. A classical oracle returns a one-bit answer (yes or no) in response to a question ("Is the 437th entry in the phone book 545-0111?" or "Is the logic function T for inputs TFTFFTFT?"). A *quantum* oracle must accept a quantum superposition of questions, and return the corresponding quantum superposition of answers.

[10] Wikipedia article *Grover's algorithm*; Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, Cambridge, 2000; and Lov K. Grover. A fast quantum mechanical algorithm for database search. 1996. URL `http://arxiv.org/abs/quant-ph/9605043`

By using a superposition it is possible to ask all possible questions of the oracle simultaneously. It might seem that a quantum oracle only needs to be consulted once to find the correct question, but measuring the qubits collapses the quantum state so the complete superposition cannot be determined. Nevertheless, Grover showed how to get the needed information from a quantum oracle in far fewer tries than would be needed classically.

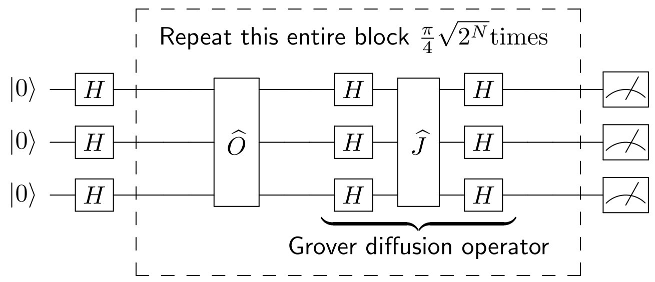

Repeat this entire block $\frac{\pi}{4}\sqrt{2^N}$times
$|0\rangle$
$|0\rangle$
$|0\rangle$
H
$\widehat{O}$
$\widehat{J}$
Grover diffusion operator

Figure 4: Quantum circuit diagram for Grover's search algorithm. Since the order in which the three Hadamard gates (on the left, for example) are applied does not change the calculation result, they are shown stacked above one another to save room in the diagram.

Grover's algorithm: details

Fig. 4 shows the quantum circuit for Grover's algorithm. The size of the database that can be searched is $D = 2^N$, where N is the number of qubits as usual. The diagram is drawn for $N = 3 \Rightarrow D = 8$. In addition to Hadamard gates, an operator $\widehat{O}$ for the quantum oracle and a special operator $\widehat{J}$ are needed. The oracle knows what the right question is to give a "yes" or T response, and it functions as follows:

- If the oracle is given one of the $D-1$ wrong questions, it returns its input state unchanged. For example, say the right question (represented as a binary number) is 110. Then, since 100 is not the right question $\widehat{O}|100\rangle = |100\rangle$.
- Conversely, if the oracle is given the right question, it returns its input multiplied by -1. Continuing the example above, this means $\widehat{O}|110\rangle = -|110\rangle$.

Therefore the quantum oracle matrix is like the identity matrix, except that it has -1 as the diagonal element for the correct question. When that question is 110, the matrix is

One can object that to write Eq. 27 we need to know in advance the answer to the search problem. For a useful application of Grover's algorithm, the oracle would be a quantum calculation that could be constructed without knowing the correct answer. For example, the oracle could implement the logic circuit of Fig. 2 using quantum gates. See Mermin pp. 88-89 for further discussion.

$$\widehat{O} \leftrightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{27}$$

since $|110\rangle$ is the seventh basis vector. The oracle $\widehat{O}$ is clearly unitary: when multiplied by its adjoint it gives the identity matrix. The operator $\widehat{J}$ in Fig. 4 is like the oracle, except that the -1 element is always in the first position:

$$\widehat{J} \leftrightarrow \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{28}$$

Here is how Grover's algorithm works. The first set of Hadamard gates in Fig. 4 creates an equal superposition of all 2^N basis states. This superposition is passed to the oracle, effectively asking all 2^N questions at the same time. The oracle flips the sign of the amplitude for the correct question. The "Grover diffusion operator" in Fig. 4 is designed to convert this *phase* difference, which is unmeasurable, into a *magnitude* difference that will show up when the qubits are measured.[11]

[11] Wikipedia article *Grover's algorithm*

It might seem better to use an oracle that encodes its output in a directly usable form,

$$\widehat{O}' \leftrightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{29}$$

If $\widehat{O}'$ is applied to a superposition of all basis states, the result is immediately the basis state for the correct question $|110\rangle$. Unfortunately, a matrix like $\widehat{O}'$ is not unitary, and so does not represent a possible operation in a quantum computer.

Programming project 3: Implement Grover's quantum search.

Implement the Grover search algorithm for $N = 3$ qubits, as diagrammed in Fig. 4. The needed matrices are given above. The optimum number of repetitions should be close to $(\pi/4)\sqrt{2^N}$ but of course it must be an integer.

- Set the oracle so the correct question is 110 as above, and with the optimum number of repetitions of the oracle plus Grover diffusion block. Run the computation many times. The measured result should be $|110\rangle$ more than 90% of the time.[12]
- Change the oracle so a different question is correct.
- Change the number of repetitions. The percentage of time that the result is correct should decrease with either more or less repetitions.

[12] Wikipedia article *Grover's algorithm*

Gates operating on more than one qubit

In Fig. 1 there are six gates that connect *two* horizontal lines. Each of these gates therefore operates simultaneously on two qubits. Consider the second two-qubit gate in Fig. 1, a "controlled NOT" or CNOT gate. This gate has a solid dot on qubit 3 (numbering qubits from 1 at the top) connected by a link to a $\oplus$ symbol on qubit 2. The NOT function changes $|0\rangle$ to $|1\rangle$, and $|1\rangle$ to $|0\rangle$. This CNOT gate applies a NOT function to qubit 2 if and only if qubit 3 is in the state $|1\rangle$. That is, qubit 3 *controls* whether or not a NOT function is applied to the qubit 2. Some other types of controlled gates can be seen in Fig. 1. In each case a solid dot on the controlling qubit is linked to a symbol indicating a conditional action on another qubit. The matrix for a CNOT gate is

$$\widehat{C}_{\text{NOT}} \leftrightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{30}$$

Exercise: Verify that this matrix carries out the CNOT function of Fig. 5(a). It helps to write the four basis vectors $|00\rangle \ldots |11\rangle$ down the side and across the top of the matrix.

Given any one-qubit gate $\widehat{U}$, it is possible to construct a two-qubit gate in which qubit 1 controls whether or not the operation $\widehat{U}$ is applied to qubit 2. This "controlled-U" gate has the symbol shown in Fig. 5(b), and has the matrix

$$\widehat{C}_U \leftrightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{0,0} & U_{0,1} \\ 0 & 0 & U_{1,0} & U_{1,1} \end{bmatrix}. \tag{31}$$

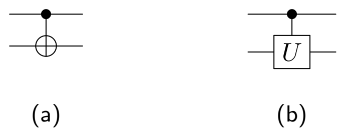


Figure 5: (a) Symbol for a CNOT gate, where the first bit is the controlling bit. (b) Symbol for the the general controlled-U gate. The CNOT is a controlled-U with U = NOT.

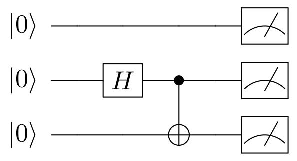
In the N=3 computer any of the three qubits can be the controlling bit and either of the remaining two qubits can be the controlled bit, so there are six possible CNOT gates. It will suffice to write out two of the six possibilities, first with qubit 2 controlling qubit 3:

$$\widehat{C}^{(2,3)}_{\text{NOT}} \leftrightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \tag{32}$$

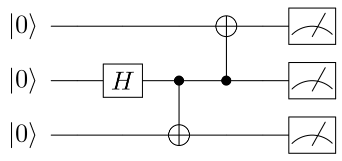
then with qubit 2 controlling qubit 1:

$$\widehat{C}^{(2,1)}_{\text{NOT}} \leftrightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{33}$$

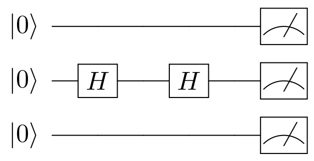
The algorithm used to generate these matrices is given in Part 2 of these handouts.



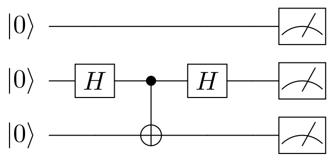
(a) Entangled state



(b) Cat state



(c) Unobserved superposition



(d) Observed superposition

Figure 6: Quantum computations using CNOT gates. See text for details.

Programming project 4: Computations using CNOT gates.

Try the quantum computations shown in Fig. 6, using the CNOT operations of Eqs. 32-33.

(a) In this diagram there is a Hadamard gate on qubit 2, before it is used to control qubit 3. The Hadamard gate puts qubit 2 in a superposition of $|0\rangle$ and $|1\rangle$ (Eq. 14). When this superposition is used to control qubit 3, that qubit will also end up in a superposition. When the computation is carried out multiple times, the output should vary randomly between $|000\rangle$ and $|011\rangle$. The measured values for qubits 2 and 3 are both random, but they are correlated with each other. This is called an *entangled* state of qubits 2 and 3.

(b) This computation creates a cat state. A Hadamard gate is used to put qubit 2 into a superposition of $|0\rangle$ and $|1\rangle$. Then two CNOTs are used to correlate qubits 3 and 1 with qubit 2. The measured results should vary randomly between $|000\rangle$ and $|111\rangle$, as they did with the cat state of Eq. 7.

(c) This duplicates an earlier calculation. First a Hadamard gate puts qubit 2 into a superposition of $|0\rangle$ and $|1\rangle$. Then a second Hadamard gate converts the superposition back into the pure state $|0\rangle$. With this computation, the measured result is always $|000\rangle$.

(d) This is like the previous diagram, but now a CNOT has been used to *observe* the state of qubit 2 when it is in a superposition of $|0\rangle$ and $|1\rangle$. Qubit 3 is flipped by the CNOT to agree with the state of qubit 2, and when qubit 3 is measured it reveals what the state of qubit 2 was between the two Hadamard gates. Observing the state of qubit 2 destroys the quantum coherence of the superposition, so the second Hadamard gate cannot put qubit 2 back into the pure state $|0\rangle$. The measured results for qubits 2 and 3 will be random and uncorrelated (equal probabilities for observing $|000\rangle$, $|001\rangle$, $|010\rangle$, $|011\rangle$).

Thinking classically the CNOT gate in Fig. 6(d) would not do anything to qubit 2 – it would only affect qubit 3. But in quantum mechanics the fact that qubit 3 is flipped by qubit 2 automatically has a big effect on qubit 2, independent of any energy scales involved. This is an example of "decoherence", discussed in more detail in Part 3 of these handouts.

Optional exercise: In Fig. 6(d) the state of qubit 2 is not really observed until the qubits are measured. Figure out how to undo the observation of qubit 2 before the qubits are measured by adding one more gate, and verify that with this modification the results are the same as for Fig. 6(c).

Figure 6(d) also illustrates a peculiarity of quantum computation that we will see later in Shor's algorithm: By using qubit 2 to control qubit 3, we affect the result of measuring qubit 2 – we can use this effect on purpose, taking the result of the computation from qubit 2.

This concludes the projects using $N = 3$ qubits. In Part 2 of these handouts the number of qubits is increased, resulting in more challenging programming tasks. This is a necessary prelude to trying Shor's algorithm, which requires $N = 7$.

Appendix: Programming hints

Binary numbers and array indices

In some computer languages (Python, C++...) array indices start from zero. If the variable `psi` is an 8-element array representing the quantum state $|\Psi\rangle$ in Eq. 5, `psi[0]`$=\ a$ and `psi[7]`$=\ h$. **In these languages to find the basis state represented by `psi[j]`, convert *j* into a binary number.** For example, $j = 6$ corresponds to $|110\rangle$.

In other computer languages (MATLAB, Mathematica...) array indices start from one. In these languages Eq. 5 implies `psi(1)`$= a$ and `psi(8)`$= h$. **In these languages to find the basis state represented by `psi(j)`, convert $j - 1$ into a binary number.** Now $j = 7$ corresponds to $|110\rangle$.

The general rule to use when doing these projects in MATLAB or Mathematica is: Add one to any number being that expresses the qubit values in binary form before using the number as an array index. Similarly, subtract one from an array index before converting it to a binary number representing the qubit values.

Convention used for numbering qubits

In a gate-array diagram like Fig. 1, qubit 1 is the top line and qubit N is the bottom line. For a basis state like $|011\rangle$, qubit 1 is the leftmost bit (0), and qubit N is the rightmost bit (1). It follows that basis states for an N-qubit register are read from top to bottom in a gate-array diagram. Thus in Fig. 1 the register is initialized to the state $|000b\rangle$.

Hints for project 1: Measurement of the N-qubit register

To carry out a quantum measurement on the state $|\Psi\rangle$ in Eq. 5,

1. Pick a random number r with $0 \leq r < 1$ (a built-in function in most languages).

2. Carry out the following steps (r is fixed during these steps; q is a new variable):

   (i) Set $q = |a|^2$. If $r < q$ the result is $|000\rangle$; otherwise proceed to the next step.

   (ii) Set $q \Leftarrow q + |b|^2$. If $r < q$ the result is $|001\rangle$; otherwise proceed to the next step.

   (iii) Set $q \Leftarrow q + |c|^2$. If $r < q$ the result is $|010\rangle$; otherwise proceed to the next step.

   ...(three steps omitted)...

   (vii) Set $q \Leftarrow q + |g|^2$. If $r < q$ the result is $|110\rangle$; otherwise the result is $|111\rangle$.

The left-arrow notation e.g. $a \Leftarrow a + 1$ is used to denote the assignment of a new value to the program variable a. In most computer languages this is written `a=a+1`.

These steps are designed to give Eq. 8: The probability of measuring $|000\rangle$ is $|a|^2$, etc. They can be programmed as above with seven IF statements, or a more elegant program can be devised that works for any N.

Hints for projects 2-4: Full quantum computations with $N = 3$ qubits

To carry out Fig. 3(a) (a Hadamard gate applied to qubit 2), multiply the column vector Ψ_k that represents $|\Psi\rangle$ on the left by the matrix for $\widehat{H}^{(2)}$ given in Eq. 22:

$$\Psi_j \Leftarrow \sum_{k=0}^{7} H^{(2)}_{j,k} \Psi_k. \tag{34}$$

This is written for Python or C++. In MATLAB or Mathematica the sum runs from 1 to 8.

This equation carries out the operation shown abstractly by Eq. 25. All of the quantum computations in this paper are done by carrying out multiple operations of the form of Eq. 34 in the order shown in the quantum gate array diagram. It will usually be more efficient to carry out the matrix multiplication of Eq. 34 using built-in matrix operations, rather than writing your own code to do this calculation.

Gates can be moved along their lines as long as the order of operations *on each line* is unchanged. So for Fig. 3(b) the three Hadamard gates could be applied in any order.

Also, recall that in a simulated quantum computer (unlike a real one), we can reuse $|\Psi\rangle$ for multiple measurements since our simulated measurements do not collapse the quantum state.

Production notes

These handouts were produced with Latex using `\documentclass{tufte-handout}` documented at

`https://tufte-latex.github.io/tufte-latex/`

The quantum circuits were drawn using the Q-circuit macro package by Bryan Eastin and Steven T. Flammia, available at

`http://physics.unm.edu/CQuIC/Qcircuit/`

The logic circuit in Fig. 2 was drawn with `\usepackage{tikz}`.

The source file for Parts 1 and 2 of these handouts is `byo1.tex` and `byo2.tex` with a shared bib file `byo12bib.bib`.