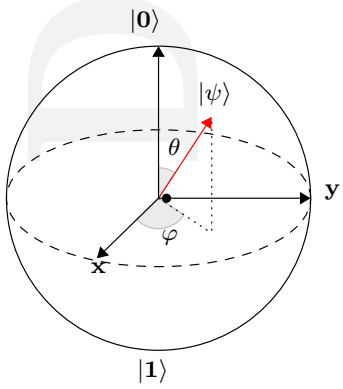
Single Qubits and Superposition

Quantum Programming is used to manipulate quantum systems. Physical quantum systems are modeled by linear algebra, and quantum programming provides a way to symbolically manipulate quantum systems by abstracting away the underlying details of the physical system the same way high level computer languages abstract away the physical operation of the hardware.

Specification for the Qubit

From a computer programming viewpoint, it is easiest to understand the qubit by starting with an abstract specification. This specification is generic, but it has the essential features of a qubit, and all quantum libraries implement all of its features in one way or another.

An abstract standard qubit stores information about a vector measured relative to the three standard axes – X, Y, and Z. Although the angle between the vector and the axes can be set in a straightforward manner, the angles cannot be read directly. Instead, there are three methods to access information held by the qubit: measureX, measureY, and measureZ. Each method returns 0 or 1 according to a probability distribution such that as any one measure's probably approaches certainty, the probabilities of the other two decrease to randomness. That is, if "measureZ" returns 0 with probability 1, both measureY and measureZ will return 0 and 1 randomly – 50/50. Having just defined an example



state, it will be used as the default initial state of the abstract standard qubit. That is, instantiating the object creates a qubit with "measureZ" returning 0 with probability 1 and the other measurements being completely random.

How this specification allows the computation of useful information will be explained in the chapters on various quantum algorithms. Why you would want these requirements in a system and what the architecture allows you to do will be explained in the chapter on simulations.

THE ABSTRACT STANDARD QUBIT CAN BE MANIPULATED by passing a 2 x 2 unitary matrix of complex numbers. This matrix will alter the probability distributions by applying a linear transformation (rotation) to the vector. These 2 x 2 matrices will be passed in as a standard set of predefined gates. These gates make up part of the specification but are too numerous to list here and are well documented in standard documentation.

THE ABSTRACT STANDARD QUBIT'S PROBABILITIES are determined by the square of the cosine between the vector and the axis being measured.

Once an abstract standard qubit is measured, its probability distributions becomes 1 on the measured axis for the determined value (either 0 or 1) and all other probability distributions are completely random. That is, any other information about the probability distributions is reset when a measurement is performed.

This means repeated measurements along the same axis return the same value. However, if you rotate the qubit, different values could be obtained.

THERE IS AN ORNAMENTAL PROPERTY called, "global phase." The global phase does not affect any of the measurements, but is useful for documenting the abstract standard qubit, as will be discussed below.

THERE IS AN ADDITIONAL PROPERTY called "entanglement" which will be defined in the next chapter. The only requirement for "entanglement" at this point is that it cannot alter the probability distributions of the qubit in any way. This requirement will remain even after entanglement is implemented.

Implementations of the Qubit

HAVING DEFINED THE REQUIREMENTS for a qubit, the actual implementation is straightforward because physicists have already worked out a model for how such a system behaves. In addition, numerous libraries exist that either simulate physical qubits or allow the manipulation of physical qubits, with the primary difference between the two being that actual qubits are "noisy." That is, because of various errors or interactions with the surrounding environment, actual qubits deviate from the exact mathematical model.

THIS BOOK PRIMARILY DEALS WITH SIMULATED qubits because the simulation allows you to "look under the hood" for how the mathematical model is behaving. This makes it much easier to understand why a qubit returns 0 in some cases and 1 in other cases. However, none of this model exists in the real world. That is, physical particles are not multiplying matrices and applying probability distributions.

VARIOUS QUANTUM COMPUTING libraries implements the "abstract standard qubit" in slightly different ways. The main implementation issue is that they generally only implement a single measurement method along the Z-axis, and this axis has known as the standard basis or computational basis. In order to measure along the other axes, you simply rotate the qubit to that axis and then perform a measurement. Defining the three axes and allowing their measurement makes it easier to write the specification for the "abstract standard qubit," but in terms of actual computations, the single measurement function works just fine.

Single Qubits and the Standard Basis

QUBITS ARE MATHEMATICALLY REPRESENTED by vectors. Depending on the type of calculation being performed, they are represented either as either column or row vectors. For mathematical formulas, the "bra-ket" notation is used with a "bra" being a row vector and a "ket" being a column vector.

The term "bra-ket" comes from the use of brackets to represent matrix operations commonly performed on quantum systems. The first half of the bracket is called the "bra" and the second is called the "ket," with the "c" being dropped.

QUANTUM SYSTEMS CAN ONLY HAVE discrete values. For a single qubit, the values are 0 and 1. A qubit in the 0 state is represented as $|0\rangle$ and a qubit in the 1 state is represented as $|1\rangle$. This representation is known as the standard basis, and 0 and 1 are represented by the column vectors ket-0 and ket-1:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

A common mistake is to represent 0 as a vector of 0s. In quantum computing, a vectors of 0s cannot exist as the Schrodinger equation states that any state that became 0 would stay that way forever.

Other bases

OTHER TYPES OF BASES include $|V\rangle$ and $|H\rangle$ for Vertical and Horizontal polarization, $|D\rangle$ and $|A\rangle$ for Diagonal and Anti-diagonal, $|\downarrow\rangle$ *and* $|\uparrow\rangle$ for Ground state and Excited, and $|+\rangle$ and $|-\rangle$ for the Hadamard basis. Any set of bases is sufficient to describe the state of a qubit, but it sometimes useful to use other bases, and the following are commonly used:

The $|+\rangle\,|-\rangle$ bases is sometimes referred to as the Hadamard basis.

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$$

$$|L\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i\,|1\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \end{pmatrix}$$

$$|R\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i\,|1\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{i}{\sqrt{2}} \end{pmatrix}$$

For physical systems based on photons, it is common to use Horizontal and Vertical bases instead of the standard bases. That is, $|H\rangle = |0\rangle$ and $|V\rangle = |1\rangle$. Similarly Diagonal/Anti-Diagonal are $|D\rangle = |+\rangle$ and $|A\rangle = |-\rangle$.

Transforming Single Qubits

QUANTUM PROGRAMMING MANIPULATES qubits by applying a transformation matrix to an existing state to arrive at a new state:

$$\left|\psi'\right\rangle = U\,|\psi\rangle$$

where $|\psi\rangle$ is the existing state and U is a unitary matrix. This matrix is referred to as a "quantum transformation", "quantum operator", "linear operator", or simply "the matrix." It is a square, normal, diaganolizable matrix of complex entries that performs a rotation on a quantum system. The study of these matrices is the entire subject of this book.

THE EXISTING STATE of a quantum system is represented by a state vector – which also called a "wave function." In the above equation, ψ is the state vector, and the transformation operation is performed to evolve the system into a new state vector. Potentially, this new state has the exact same value as the old state.

Some quantum programming frameworks, such as QisKit, call it a state vector, while others call it a wave function. Typically, physics centered approaches call it a wave function and information centered approaches call it a state vector.

An example of a simple rotation is the big-flip operation. It transforms $|0\rangle$ into $|1\rangle$, similar to a classical NOT gate. Transformations on a single qubit are represented by 2x2 matrices. In order to flip a qubit from 0 to 1 or vice-versa, apply a transformation as follows:

$$U|\psi\rangle = X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

IN QUANTUM PROGRAMMING, you call functions that build and manipulate the matrix. Ultimately, all quantum programming is constructing these matrix transformations to move from one quantum state to another.

Single Qubit Transformations in Bra-Ket

Bra-ket notation is useful in describing basic quantum transformations. Above, the X gate was used to map the $|0\rangle$ state into $|1\rangle$, formally $|0\rangle \mapsto |1\rangle$. The X gate also maps $|1\rangle$ into $|0\rangle$, $|1\rangle \mapsto |0\rangle$. Above it was simply presented by fiat, but you can derive the gate using Bra-ket notation by projecting each existing basis state onto the desired basis state:

$$X = |0\rangle\langle 1| + |1\rangle\langle 0| =$$

$$\begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} =$$

$$\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

The Y gate maps $|0\rangle \mapsto |1\rangle$ with an amplitude of $\sqrt{1} = -i$ and $|1\rangle \mapsto |0\rangle$ with amplitude $\sqrt{-1} = i$. In bra-ket notation, and factoring out the coefficient to the front of the term, this is:

$$Y = -i|0\rangle\langle 1| + i|1\rangle\langle 0| =$$

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

ANY SINGLE QUBIT TRANSFORMATION can be represented as the combination of projections of each basis vector onto the other basis vectors:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = a\,|0\rangle\,\langle 0| + b\,|0\rangle\,\langle 1| + c\,|1\rangle\,\langle 0| + d\,|1\rangle\,\langle 1|$$

With the additional qualifications that the transformation matrix be unitary, this can be seen as a mapping input values to probabilistic outputs. In order to be unitary, the conjugate transpose of the matrix must be its inverse.

$$UU^\dagger = I$$

For the generic one qubit mapping this is:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} a & c \\ b & d \end{pmatrix} = \begin{pmatrix} a^2 + b^2 & ac + bd \\ ac + bd & c^2 + d^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

IN PRACTICAL TERMS, this means that the sum of the squares of any probabilistic mapping must be one and $ac = -bd$. The latter requirement of $ac = -bd$ is handled using "relative phase," and this concept will be covered at length shortly.

AS AN EXAMPLE, to transform an input of $|0\rangle$ onto either $|0\rangle$ or $|1\rangle$ with 50/50 probabilities, the values of a and b would be $\frac{1}{\sqrt{2}}$, as the sum of squares of these probabilities must equal one.

And to transform an input value of $|1\rangle$ on $|0\rangle$ or $|1\rangle$ with 50/50 probabilities, the values of c and d would also be $\frac{1}{\sqrt{2}}$. Here, we will apply a "relative phase" of -1 to d to keep the matrix unitary, with an explanation of what they means coming later. The transformation is:

$$\frac{1}{\sqrt{2}}|0\rangle\langle 0| + \frac{1}{\sqrt{2}}|0\rangle\langle 1| + \frac{1}{\sqrt{2}}|1\rangle\langle 0| - \frac{1}{\sqrt{2}}|1\rangle\langle 1| =$$

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

The above is known as the Hadamard transformation and it is used extensively in quantum programming.

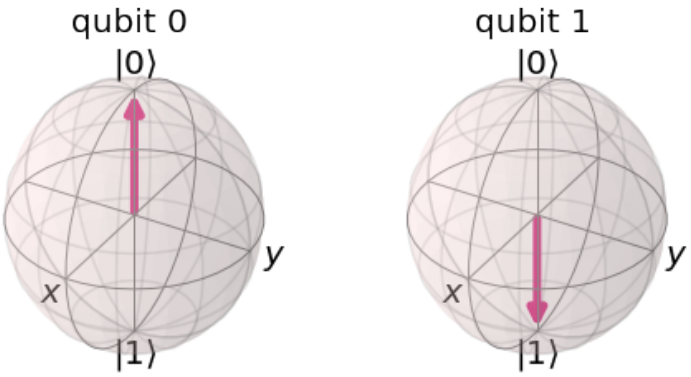
Visual Representation of a Qubit

Qubits can be visually represented as vectors pointing outwards to a sphere. Such representations are called Bloch sphere representations and they are useful to see various rotations being performed.

The standard basis is visually represented as $|0\rangle$ extending up and $|1\rangle$ extending down and can be visualized using Qiskit:

Qubits are initialized to 0. Flip the second qubit to 1 and display. The qubit numbers are zero based.

```
circuit = QuantumCircuit(2)
circuit.x(1)
plot_bloch_multivector(lq.execute_state_vector(circuit))
```



BLOCH SPHERE REPRESENTATIONS are useful to see rotations, but they are also a source for confusion because the two basis vectors $|0\rangle$ and $|1\rangle$ are parallel. This the projection onto the sphere does not preserve the angles between the vectors. Another point of confusion is that negative numbers are not represented. Given the various limitations of the Bloch sphere representation, their use will be limited to explaining basic single qubit rotations. However, some quantum simulators, such as QCEngine, use Bloch spheres and various circle notions to visually represent all manners of transformations.

Bloch spheres are just a tool to visualize rotations. They do not contain all the information encoded in a qubit.

Single qubits in superposition

A SUPERPOSITION is a combination of states along a given basis with a probabilistic chance of either state being measured.

The state of the qubit in superposition is represented by the sum of the proportional probabilities of obtaining the given basis vector should the qubit be measured. This is represented by the state vector as $ket\psi = \alpha\,|0\rangle + \beta\,|1\rangle$ where α and β are complex values such at $\|\alpha\|^2 + \|\beta\|^2 = 1$. The values α and β are not the probabilities themselves but rather the square roots of the probabilities.

Technically, every qubit is in superposition of some basis at all times. But it is common to refer to the standard basis for superposition unless it is explicitly stated otherwise.

Using the spherical representation, the state vector is

$$|\psi\rangle = cos(\frac{\theta}{2})|0\rangle + e^{\varphi i} sin(\frac{\theta}{2})|1\rangle$$

This models complies with the requirement that as the state vector moves closer to certainty along any one axis, the other two are completely random.

WHEN YOU TAKE A MEASUREMENT, a qubit will return exactly one state with the probability of each state being returned defined by the state vector. Once a qubit is measured, any other information inside the qubit is lost and the qubit remains in its new state until it is acted upon. That is, if you measure a qubit multiple times, it will continue to return the same result unless you rotate it or perform some other operation on it.

Whether the qubit is always in one state or the other or whether it moves into a specific state when it is measured is an interesting question but not relevant to quantum programming at this point.

When a quantum computer runs multiple "shots" and measures the same qubit it gives different outcomes. This is because the computer is resetting the qubit each time, rotating it according to the gate instructions and then performing a new measurement.

MANY TEXTS ILLUSTRATE SUPERPOSITION with the Hadamard transformation. The Hadamard transformation will indeed create superposition, but the superposition it creates is rather complex, and students frequently confuse "Hadamard" with "superposition" because of this early association. The Hadamard gate is an extremely important gate, and it will be used throughout this book, but understanding superposition is more straight forward by looking at a simple rotation.

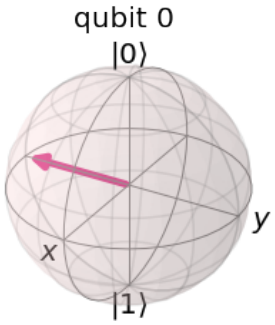
Rotating the $|0\rangle$ vector 90°around the X-axis results in a superposition in the standard basis:

```
circuit = QuantumCircuit(1)
circuit.rx(np.pi/2, 0)
Latex(lq.show_state_vector(circuit))
```

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + -\frac{1}{\sqrt{2}}i|1\rangle$$

The Bloch sphere shows the rotated vector in the middle between $|0\rangle$ and $|1\rangle$.

There is nothing special about a 50/50 superposition, and you can set any probability you wish, as the exercises show.



The probabilities computed from the state vector are:

$$\|\frac{1}{\sqrt{2}}\|^2 = \frac{1}{2}|0\rangle \quad \text{and} \quad \|-\frac{1}{\sqrt{2}}i\|^2 = \frac{1}{2}|1\rangle$$

This creates a superposition state with the probabilities determined by the angle of rotation. In this case, the $\frac{\pi}{2}$ parameter corresponds with a probability of $cos(\frac{\pi}{4}) = \frac{1}{2}$. The angle is divided by two according to the formula for the rotation matrix which is given below.

Measurement

QUANTUM COMPUTERS TYPICALLY MEASURE in the standard basis, which is also called the computational basis. You can measure in a different basis by rotating the qubits into that basis and performing a measurement. Most systems only provide a single "computational basis," leaving it to the programmer to perform a uniform rotation of all the qubits if a different measurement basis is desired.

Many quantum programming problems are significantly easier when computed in other bases and the bases of any number of qubits can be changed simply by rotating them. The state vector is also referred to as the wave function.

THE STATE VECTOR for a system is the transformation matrix applied to the initial state of the system. For quantum computers, qubits are typically initialized as 0, so for a single qubit system, the state vector is:

$$|\psi\rangle = U\,|0\rangle$$

If you have not applied any transformations, U is the identity matrix and the state of the system is $|0\rangle$.

IN A REAL QUANTUM COMPUTER, there is no way to measure the state vector. All of the manipulations are based on a mathematical model used to predict how particles act. The only thing you can determine is a measurement of the system. That is, the hardware will query the state of the qubit and the qubit will return exactly one value – either a 0 or a 1. At that point, the state vector of the system is back to $|0\rangle$ or $|1\rangle$. Any superposition or other information in the system is removed and the state vector is reduced to a single value.

Mathematically, the qubit has taken on one of the eigenstates of the system. Eigenvalues and Eigenvectors will be covered in more detail, but their use explains what is meant by the phrase "the wave function collapses." There is no "collapse" of any function. In an eigenstate, a matrix term can be represented by a scalar and thus some people refer to this as a "collapse."

An Eigenvector is also referred to as an Eigenstate.

EIGENVECTORS correspond to Eigenvalues which are scalars that simplify a matrix. Their definition is:

$$A\mathbf{v} = \lambda\mathbf{v}$$

That is, if you have a vector and apply a transformation A, if v is an eigenvector, the product can be simplified down to multiplying

v by a scalar product. For quantum system, the vector is always an eigenvector. For single qubits, no matter how many transformations you apply, no matter how complex your transformation matrix becomes, the end result is always either 0 or 1. Prior to measurement, your matrix and the corresponding state vector may include multiple possibilities. Once it is measured, only one possibility exists.

If working in the +/- basis, it to reduced to multiplying by +1 or -1.

MEASUREMENT IS NOT THE ONLY WAY to reduce the state vector. Any transformation that returns the state vector to only one possible value has the same effect as a measurement – that is, the wave function can be represented by multiplying a single basis of the state vector by an amplitude of 1, with the other basis being 0. You can create a superposition:

```
circuit = QuantumCircuit(1,1)
circuit.rx(pi/2, 0)
Latex(lq.show_state_vector(circuit, label='\psi_{Rx}'))
```

$$|\psi_{Rx}\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}i|1\rangle$$

And "collapse" the wave function:

```
circuit.rx(-pi/2, 0)
Latex(lq.show_state_vector(circuit, label='\psi_{Rx-Rx}'))
```

$$|\psi_{Rx-Rx}\rangle = 1|0\rangle$$

Which is the same as a measurement:

```
circuit = QuantumCircuit(1,1)
circuit.rx(pi/2, 0)
circuit.measure(0,0)
Latex(lq.show_state_vector(circuit, label='\psi_{Meas}'))
```

$$|\psi_{Meas}\rangle = 1|0\rangle$$

It should be noted that the above state vector is randomly assigned. If you run the code, it will give different state vectors depending on the random assignment of the measurement. But the state vector will always be either $|0\rangle$ or $|1\rangle$. Here, they have

equal probability, so if you don't get that state vector on your first try, just re-execute the code and eventually the state vector of $|0\rangle$ will show up.

THERE IS SOME PHILOSOPHICAL debate about whether a measurement is different from a rotation that achieves the same state vector. In quantum programming, if two state vectors are identical, it is assumed they represent identical states, and any philosophical differences between the two states will be ignored.

Gate Composition

CONSTRUCTING GATES to perform specific operations is a key skill for quantum programming. Single qubits operations are represented by 2 x 2 matrices of complex entries. No matter how many gates you apply to a single qubit, they all combine into four complex entries in the transformation matrix. If you know the gates to apply, computing the matrix is trivial, and Qiskit or any other quantum programming interface will compute it for you.

Alternately, you can simply multiply them together:

```
lq.pauliX@lq.pauliY@lq.hadamard
```

Or let Qiskit do it for you. The gates are applied from right-to-left:

```
circuit = QuantumCircuit(1)
circuit.h(0)
circuit.y(0)
circuit.x(0)
Latex(lq.show_me_the_matrix(circuit))
```

Both produce the same matrix:

$$\frac{1}{\sqrt{2}}\begin{pmatrix} i & i \\ -i & i \end{pmatrix}$$

THE CHALLENGE is when you are given a matrix and you wish to put together a sequence of gates to construct it. For single

The final optimization step depends on the physical quantum hardware. Different quantum computers handle gates differently and produce higher error rates. Your goal is to minimize those gates and chose a decomposition system which best optimizes your hardware.

qubits, this task is straight forward, but it gets more complicated as the number of qubits increases. The reason you will be starting with a matrix is that one way to address programs tractable on quantum computers is to first determine a set of states that solves your problem, then determine the eigenvectors and eigenvalues for those states and use those to construct a matrix. Once you have the matrix, you can decompose it into gates using a formula. Once you have the mechanical decomposition of the gates, determine if it can be optimized.

As will be seen in several of the chapters, mechanical decompositions give complex and difficult to work with matrices. As you get experience, you will be able to spot more elegant solutions.

The formula for single qubit decomposition is:

$$e^{i\alpha}\begin{pmatrix} e^{-i\beta/2} & 0 \\ 0 & e^{i\beta/2} \end{pmatrix}\begin{pmatrix} \cos\frac{\gamma}{2} & -\sin\frac{\gamma}{2} \\ \sin\frac{\gamma}{2} & \cos\frac{\gamma}{2} \end{pmatrix}\begin{pmatrix} e^{-i\delta/2} & 0 \\ 0 & e^{i\delta/2} \end{pmatrix}$$

The above can further be factored for unitary gate operations as:

$$e^{i\alpha}Rz(\beta)Ry(\gamma)Rz(\delta).$$

The term $e^{i\alpha}$ represents a global phase shift for which there is no standard gate for an arbitrary rotation because for measurement purposes, the global phase is irrelevant.

The Qiskit Rz gate phase causes Qiskit implementations to differ by a global phase from other implementations or from matrices published in academic texts. The global phase difference has no measurable result, but if you want your phases to match with a publication, you need to use adjust the global phase.

The current IBM quantum computers implement U1, U2, U3 and CNOT as basis gates. By incorporating the global shift into the U1 gate, it turns the gate into a pure phase shift which is handled by just changing the timing of the measuring device. This reduces errors when compared to physically rotating the qubit.

The mathematics of the Rz gate is covered in an appendix. For now, it will just be noted that the Qiskit implementation of Rz differs from others by a global phase. For instance on PyQuil:

```
p = Program(RZ(pi, 0))
Latex(a_to_l(ut.program_unitary(p, 1)))
```

Shows:

$$\begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix}$$

While on Qiskit:

```
circuit = QuantumCircuit(1)
circuit.rz(pi, 0)
Latex(lq.show_me_the_matrix(circuit))
```

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

As the two states differ only by a global phase, it is not possible to measure a difference.

Mechanical Composition

To reproduce a specific matrix, you can mathematically solve for α, β, γ, and δ. Fortunately, Qiskit provides a function to find β, γ, and δ, and we write a wrapper function to get those values and then compute α.

```
def decompose_single(unitary_matrix):
    (theta, phi, lamb) = twoq.euler_angles_1q(unitary_matrix)
    qr = QuantumRegister(1)
    qc = QuantumCircuit(qr)
    qc.append(rrz_gate(lamb), [qr[0]])
    qc.ry(theta, qr[0])
    qc.append(rrz_gate(phi), [qr[0]])
    new = what_is_the_matrix(qc)
    alpha = get_global_phase(unitary_matrix, new)
    print('alpha= {}, beta= {}, gamma= {}, delta={}'
          .format(format_rotation(alpha),
                  format_rotation(phi),
                  format_rotation(theta),
                  format_rotation(lamb)))
```

The euler_angles_1q computes the rotations angles except for global phase. Although it does compute a "phase," that is just to solve for the other variables and it does not always equal the global phase for the angles returned. This functions first constructs a matrix for the angles computed by Qiskit then solves for the global phase by inverting that matrix and multiplying it by the original matrix. This will produce the global phase values either in the left-right diagonal or the right-left diagonal.

The global phase is computed by determining the difference between the two matrices:

```
def get_global_phase(original, new):
    if np.allclose(original, new):
        alpha = 0
    else:
        m_factor = original @ np.linalg.inv(new)
        if not np.isclose(m_factor[0, 0], 0):
            factor = phase(m_factor[0, 0])
        else:
            factor = phase(m_factor[0, 1])

        if np.allclose(original,
                       (np.exp(imag * factor)) * new):
            alpha = factor
        else:
```

```
            raise ValueError('New Matrix not equal to old ')
    return alpha
```

THE GLOBAL GATE is handled by a custom gate. Although you can set up parameterized gates, the parameters cannot be manipulated and it is troublesome to bind the parameters. For the custom Rz gate, we need to divide by two and take the negative, so we cannot use parameterized gates unless we define multiple parameters. It is easiest to just call a function that builds the sub-circuit with the parameter and add it as a custom gate. These custom gates are defined in learn_quantum. The global gate corresponds with $e^{i\alpha}$

```
def global_gate(alpha):
    name = 'G \n' + format_rotation(alpha)
    sub_global = QuantumCircuit(1, name=name)
    sub_global.rz(alpha, 0)
    sub_global.y(0)
    sub_global.rz(alpha, 0)
    sub_global.y(0)
    return sub_global.to_instruction()
```

The RRz gate is a Rz rotation without the global phase adjustment:

$$RRz = \begin{pmatrix} e^{-i\beta/2} & 0 \\ 0 & e^{i\beta/2} \end{pmatrix} =$$

$$e^{i\beta/2}Rz = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\beta} \end{pmatrix}$$

```
def rrz_gate(beta):
    name = 'RRz \n' + format_rotation(beta)
    sub_rrz = QuantumCircuit(1, name=name)
    sub_rrz.rz(beta / 2, 0)
    sub_rrz.x(0)
    sub_rrz.rz(-beta / 2, 0)
    sub_rrz.x(0)
    return sub_rrz.to_instruction()
```

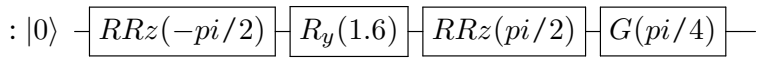
You can put the custom gates into a circuit using the append method:

```
qr = QuantumRegister(1)
custom = QuantumCircuit(qr)

alpha = pi/4
beta = pi/2
gamma = pi/2
delta = -pi/2

custom.append(lq.rrz_gate(delta), [qr[0]])
custom.ry(gamma, qr[0])
custom.append(lq.rrz_gate(beta), [qr[0]])
custom.append(lq.global_gate(alpha), [qr[0]])

custom.draw(output='mpl')
```



Exercise 1

Construct a single qubit that flips the bit value only 25% of the time. That is, with an input of $|0\rangle$, 75% of the time it will output $|0\rangle$ and 25% it outputs $|1\rangle$. Similarly, it flips $|1\rangle$ 25% of the time.

Answer: Working through mechanically, bra-ket notation can break it down because all the inputs and outputs are specified. With input 0, output is 0 with probability amplitude $\sqrt{\frac{3}{4}}$. With input 0, output is 1 with probability amplitude $\sqrt{\frac{1}{4}}$. And the reverse for inputs of 1. Breaking this out with bra-ket notation yields:

The amplitudes are the square roots of the probabilities of those states.

$$\sqrt{\frac{3}{4}}\,|0\rangle\langle 0| \;-\; \sqrt{\frac{1}{4}}\,|0\rangle\langle 1| \;+\; \sqrt{\frac{1}{4}}\,|1\rangle\langle 0| \;+\; \sqrt{\frac{3}{4}}\,|1\rangle\langle 1|$$

THE NEGATIVE SIGN was applied to the second term because that corresponds to an *Ry* rotation which is used to illustrate the mechanical decomposition in the next step. However, there is no "right" way to apply it because this problem is under specified. That is, the only requirements for this problem is that an initial

If you can spot that this transformation would be an *Ry* when the phase is applied to a certain term, it makes for an easy decomposition. If you can't spot it, call the decompose function.

qubit in state 0 be mapped on to 0 75% of the time and 1 25% of the time. Had mappings for another axis been provided, there would only be one way to assign the relative phase.

For this problem, the negative can be applied to any term – all of which result in the same probability distribution for this single qubit as it is only ever measured in the standard basis. The negative sign needs to be applied to one term so that the matrix is unitary.

In this simple case, it does not matter where you put the negative sign. However, if you applied additional transformations that interact with the phase (such as Hadamard), the system would return different values.

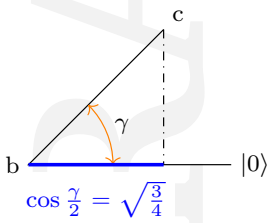
Returning to the formula and expanding the ket-bras:

$$\begin{pmatrix} \sqrt{\frac{3}{4}} & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & -\sqrt{\frac{1}{4}} \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \sqrt{\frac{1}{4}} & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{\frac{3}{4}} \end{pmatrix}$$

$$= \begin{pmatrix} \sqrt{\frac{3}{4}} & -\sqrt{\frac{1}{4}} \\ \sqrt{\frac{1}{4}} & \sqrt{\frac{3}{4}} \end{pmatrix}$$

This example is simple, but if the decomposition was not obvious, you can pass the matrix to the decompose_single function to decompose the gates:

```
u = sqrt(3/4)*ket('0')@bra('0') -
 ↪  sqrt(1/4)*ket('0')@bra('1') +
 ↪  sqrt(1/4)*ket('1')@bra('0') +
 ↪  sqrt(3/4)*ket('1')@bra('1')
lq.decompose_single(u)
Output:
alpha= 0, beta= 0, gamma= pi/3, delta=0
```



Which gives the following decomposition:

$$\begin{pmatrix} \cos\frac{\gamma}{2} & -\sin\frac{\gamma}{2} \\ \sin\frac{\gamma}{2} & \cos\frac{\gamma}{2} \end{pmatrix} = \begin{pmatrix} \sqrt{\frac{3}{4}} & -\sqrt{\frac{1}{4}} \\ \sqrt{\frac{1}{4}} & \sqrt{\frac{3}{4}} \end{pmatrix}$$

For an initial state of $|0\rangle$, the transformation to the new state vector needs to give:

$$\psi' = \sqrt{\frac{3}{4}}|0\rangle + \sqrt{\frac{1}{4}}|1\rangle$$

Applying the matrix derived above verifies that the transformation is correct:

$$\psi' = \begin{pmatrix} \sqrt{\frac{3}{4}} & -\sqrt{\frac{1}{4}} \\ \sqrt{\frac{1}{4}} & \sqrt{\frac{3}{4}} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \sqrt{\frac{3}{4}} \\ \sqrt{\frac{1}{4}} \end{pmatrix}$$

$$= \sqrt{\frac{3}{4}} |0\rangle + \sqrt{\frac{1}{4}} |1\rangle$$

Whether you used trigonometry or the decompose_single function, the result is a R_y rotation of $\pi/3$. The following is a circuit with a single Ry gate with a $\pi/3$ rotation:

An Rx rotation will work as well but Rx rotations are not part of the mechanical decomposition.

```
qr = QuantumRegister(1)
cr = ClassicalRegister(1)
circuit = QuantumCircuit(qr, cr)
circuit.ry(np.pi/3, qr[0])
m = QuantumCircuit(qr, cr)
m.measure(qr, cr)
answers = lq.execute_simulated(circuit+m ,1000,
                               seed_simulator=12345)
lq.print_reverse_results(answers)
```

Reversed: [('0', 746), ('1', 254)]

```
Latex(lq.show_me_the_matrix(circuit))
```

$$\begin{pmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{pmatrix}$$

Exercise 2

Verify that different relative phases of a the qubit in Exercise 1 causes different probability distributions when measured against the X axis (apply a Hadamard transformation prior to measurement).

Answer: Switch the negative sign to the last term. You can verify that this still gives the same probability distribution when measured in the standard basis. But it gives a very different probability distributions from the Ry rotation when they are both measured in the Hadamard basis. This is because the "relative

phase" of the systems is different, in this case opposite. As such, when the system is rotated into a basis that reflects the phase, the results returned will be different (in this case opposite).

Here, the system in Exercise 1 was under specified. That is, in our initial setup, the only thing required was a single axis measurement split 75/25. This gave several options for how to handle the phase — none of which affected the outcome of the simple exercise. For most quantum programs, the phase is a major component of the computation and it needs to be properly defined.

Move the negative sign to the last term and decompose:

This exercise introduces lq.decompose_single_qiskit(u) – which decomposes a single qubit matrix into a single U3 gate. The decompose_single function would work as well, but would require multiple gates. To keep the example as simple as possible, it is easier to use the qiskit single gate decomposition.

```
u = sqrt(3/4)*ket('0')@bra('0') +
 ↪  sqrt(1/4)*ket('0')@bra('1') +
 ↪  sqrt(1/4)*ket('1')@bra('0') -
 ↪  sqrt(3/4)*ket('1')@bra('1')
lq.decompose_single_qiskit(u)
```

Output: theta= pi/3, phi= 0, lambda= pi, phase=0

```
qc = QuantumCircuit(1)
qc.u3(pi/3,0, pi, 0)
Latex(lq.show_me_the_matrix(qc, label='U3(\pi/3,0,\pi)'))
```

$$R_{u3} = \begin{pmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{\sqrt{3}}{2} \end{pmatrix}$$

```
qc = QuantumCircuit(1)
qc.ry(pi/3, 0)
Latex(lq.show_me_the_matrix(qc, label='R_y'))
```

$$R_y = \begin{pmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{pmatrix}$$

```
qr = QuantumRegister(1)
cr = ClassicalRegister(1)
rel_phase = QuantumCircuit(qr, cr)
rel_phase.u3(-pi/3,0, pi, qr[0])
rel_phase.h(qr[0])

m_rel = QuantumCircuit(qr , cr)
```

```
m_rel.measure(qr[0], cr[0])

answers = lq.execute_simulated(rel_phase + m_rel ,1000,
                               seed_simulator=12345)
lq.print_reverse_results(answers)
```

Output: Reversed: [('0', 66), ('1', 934)]

```
qr = QuantumRegister(1)
cr = ClassicalRegister(1)
initial_ry = QuantumCircuit(qr, cr)
initial_ry.ry(np.pi/3, qr[0])
initial_ry.h(qr[0])

m_ry = QuantumCircuit(qr , cr)
m_ry.measure(qr[0], cr[0])

answers = lq.execute_simulated(initial_ry + m_ry ,1000,
                               seed_simulator=12345)
lq.print_reverse_results(answers)
```

Output: Reversed: [('0', 945), ('1', 55)]

And you can see the state vector are different:

```
Latex(lq.show_state_vector(rel_phase, label='\psi_{u3}') +
↪  lq.llf
     + lq.show_state_vector(initial_ry, label='\psi_{Ry}'))
```

$$|\psi_{u3}\rangle = 0.2588|0\rangle + 0.9659|1\rangle$$
$$|\psi_{Ry}\rangle = 0.9659|0\rangle + 0.2588|1\rangle$$

Exercise 3

Show that there is no way to "reset" a qubit using standard transformations. That is, there is no single qubit gate that takes either a 0 or 1 as input and always outputs 0.

Answer: Breaking down the inputs and outputs with bra-ket:

$$1\,|0\rangle\langle 0| \;+\; 0\,|0\rangle\langle 1| \;+\; 1\,|1\rangle\langle 0| \;+\; 0\,|1\rangle\langle 1|$$
$$= \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

Here, there is no way to assign any relative phase to make a unitary matrix, and you can verify that a unitary matrix is impossible. A unitary matrix requires:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} a & c \\ b & d \end{pmatrix} = \begin{pmatrix} a^2+b^2 & ac+bd \\ ac+bd & c^2+d^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Here:

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

Switching the signs or changing the amplitudes will not remedy the problem. As such, there is no way to arbitrarily "reset" a qubit using quantum transformations.

OF COURSE, QUANTUM COMPUTERS reset qubits. Every time you run a quantum program, the qubits get reset to zero. However, this does not happen within the system itself. Instead, the system first measures the qubits and then uses that measurement to switch the qubit to 0 if necessary.

Because of this inability to reset qubits within the system, many of the quantum sub-algorithms you develop need to "uncompute" the qubits back to zero when they finish with them so that the next part of your algorithm starts with a known and consistent state.

As you develop quantum sub-algorithms, "uncomputing" qubits will be an important step. By reversing the calculations, the qubit is reverted to its original state – 0.

Exercise 4

Use bra-ket notation to make a gate that maps $|0\rangle$ onto $|+\rangle$ and $|1\rangle$ onto $|-\rangle$.

Answer: The mapping is from 0 to + with 100% and 1 to - with 100%:

$$1\,|0\rangle\langle+| \quad + \quad 1\,|1\rangle\langle-| =$$

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 \end{pmatrix}\begin{pmatrix} 0 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

And in Python:

```
ket('0')@bra('+')  + ket('1')@bra('-')
```

Which is the Hadamard transformation – this is just another way to construct it.

Exercise 5

Decompose the Hadamard transformation into Rz and Ry gates.

Answer: You can do a mechanical breakdown:

```
lq.decompose_single(lq.hadamard)
Output:
alpha= pi/2, beta= 0, gamma= pi/2, delta=pi
```

Here, the academic breakdown requires a global phase – which Qiskit's Rz provides. As such, it can be constrcuted from Qiskit Rz and Ry gates with no need for the global phase. The matrices are multiply from right to left, so the gate are applied "backwards."

```
circuit = QuantumCircuit(1)
circuit.rz(pi, 0)
circuit.ry(pi/2, 0)
Latex(lq.show_me_the_matrix(circuit, label='Hadamard'))
```

Another way to break it down is to use Qiskit's internal mapping - but this only maps onto the U gates.

```
circuit = QuantumCircuit(1)
circuit.h(0)
circuit.decompose().draw()
```

$$q3_0 : |0\rangle \; \boxed{U_2\,(0, pi)}$$

And you can verify the matrices are the same:

$$U_2(0,\pi) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Exercise 6

What is the Hadamard rotation with a global phase shift of $\pi/4$?

Answer:

```
qr = QuantumRegister(1)
circuit = QuantumCircuit(qr)
circuit.append(lq.global_gate(pi/4), [qr[0]])
circuit.h(qr[0])
Latex(lq.show_me_the_matrix(circuit, label='Hadamard'))
```

Although it looks very different, this matrix is functionally the same as any other representation of the Hadamard transformation.

$$Hadamard = \frac{1}{2}\begin{pmatrix} 1+i & 1+i \\ 1+i & -1-i \end{pmatrix}$$

Exercise 7

Decompose the $\sqrt{NOT}$ gate. The $\sqrt{NOT}$ is a a single qubit gate that applying twice gives the same as an X gate.

Answer: The matrix is

$$\sqrt{NOT} = \frac{1}{2}\begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix}$$

You can find the mechanical decomposition by calling the decompose function.

```
srn = np.array([[0.5+0.5j, 0.5-0.5j],
       [0.5-0.5j, 0.5+0.5j]])
lq.decompose_single(srn)
```

Output: alpha= pi/4, beta= -pi/2, gamma= pi/2, delta=pi/2

Use the above terms to construct the mechanical breakdown:

```
qr = QuantumRegister(1)
circuit = QuantumCircuit(qr)

alpha = pi/4
beta = -pi/2
gamma = pi/2
delta = pi/2
```

```
circuit.append(lq.rrz_gate(delta), [qr[0]])
circuit.ry(gamma, qr)
circuit.append(lq.rrz_gate(beta), [qr[0]])
circuit.append(lq.global_gate(pi/4), [qr[0]])

Latex(lq.show_me_the_matrix(circuit, label='\sqrt{NOT}'))
```

$$\sqrt{NOT} = \frac{1}{2}\begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix}$$

While the above gives you an implementation of the transformation, the number of gates makes it inefficient to implement. You can combine various gates and further optimize the mechanical solution. However, if you look at what it is really doing, it is nothing more than a phase shift in the Hadamard basis:

```
qc_srn = QuantumCircuit(1)
qc_srn.h(0)
qc_srn.s(0)
qc_srn.h(0)
Latex(lq.show_me_the_matrix(qc_srn, label='\sqrt{NOT}'))
```

The $\sqrt{NOT}$ transformation is typically represented as above, which includes a global phase shift. Without the global phase, it is:

```
circuit.u3(pi/2, -pi/2, pi/2, qr[0])
```

$$\sqrt{NOT} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$$

And you can verify that any two matrices are equal up to the global phase:

```
mat = lq.what_is_the_matrix(qc_srn)
mat2 = lq.what_is_the_matrix(qc_no_global)
import qiskit.quantum_info.operators.predicates as pred
pred.matrix_equal(mat, mat2, ignore_phase=True)
```