PULSEE - Quantum Computing Module

Lucas Brito

December 13, 2021

Abstract

I present a quantum computing module for the Program for simULation of Spin Ensemble Evolution (PULSEE). For the past few decades, quantum computing and quantum information science have remained active and fertile fields, and the prospect of expediting certain computational tasks using quantum algorithms is expected to continue to motivate research in the field for many more years. Encouraged by this persistent and widespread interest in quantum computing—and aiming to continue PULSEE's mission to bridge condensed matter theory and experiment through simulations of spin evolution phenomena—I have implemented the fundamental components of quantum circuits in the form of a new PULSEE module titled `Quantum_Computing`.

Contents

1 Theory

1.1 Fundamentals

Where the fundamental building block of classical computing is the bit, the quantum computing analogue is the *qu*bit. Scherer (2019) (which we will henceforth use as a reference for rigorous definitions of the objects the presented software implements) defines a **qubit** as "a quantum mechanical system described by a two-dimensional Hilbert space $\mathbb{H}$" wherein "we select [an orthonormal basis $\{|0\rangle, |1\rangle\}$ and an observable represented by a self-adjoint operator denoted by σ_z" such that

$$\sigma_z |0\rangle = 1 \cdot |0\rangle \quad \text{and} \quad \sigma_z |1\rangle = -1 \cdot |1\rangle ;$$

i.e., informally, we may define a qubit to be the eigenket expansion of some $|\alpha\rangle$ in the two-dimensional basis $|0\rangle$, $|1\rangle$, where $|0\rangle$ and $|1\rangle$ are eigenkets of some (implicitly defined) observable with eigenvalues $+1$, -1 respectively.[3] As always, there is freedom in what basis one chooses; as such, we term the basis corresponding to the aforementioned observables the **computational basis**. The Hilbert space spanned by $|0\rangle$, and $|1\rangle$ is called a **qubit space**. Note that we are free to choose any convenient observable provided the above requirements are met; spin- $\frac{1}{2}$ systems or polarized light are two common choices.

A crucial feature of quantum computing is the capability for quantum circuits (and thus quantum algorithms) to leverage entangled states. In order to properly describe such states—in particular, in order to operate on both states at once—we define **tensor products** of Hilbert spaces; e.g., if $\mathbb{H}^A$ and $\mathbb{H}^B$ are Hilbert spaces with dimensions n_A and n_B respectively, the tensor product $\mathbb{H}^A \otimes \mathbb{H}^B$, one can show, is itself a Hilbert space with dimension $n_A n_B$. The elements of $\mathbb{H}^A \otimes \mathbb{H}^B$ are tensor products between state kets; e.g., $|\phi \otimes \psi\rangle$ with $|\phi\rangle \in \mathbb{H}^A$ and $|\phi\rangle \in \mathbb{H}^B$. The formal definition of such tensor produts is quite technical and thus I refer the reader to Scharer §3.2 for a rigorous treatment; the two results of interest from this treatment are: $\mathbb{H}^A \otimes \mathbb{H}^B$ is isomorphic to $\mathbb{C}^{n_A n_B}$ (i.e., we are permitted matrix representations of elements of this tensor product), and the basis kets of this Hilbert space tensor product may be written, as column vectors,

$$|e_a \otimes f_b\rangle = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}^T$$

where $|e_a\rangle$ and $|f_b\rangle$ are orthonormal basis kets for $\mathbb{H}^A$ and $\mathbb{H}^B$ respectively, and, importantly, the one appears in the $[(a-1)n_B + b]$-th entry. One can think of this column vector as having $\dim \mathbb{H}^A$ partitions of $\dim \mathbb{H}^B$ rows such that the partition we are in is determined by a and the "slot" in that partition is determined by

b. To illustrate this, consider the case $\dim \mathbb{H}^A = 2$ and $\dim \mathbb{H}^B = 3$. We have

$$\begin{aligned}
|e_1 \otimes f_1\rangle &= \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^T \\
|e_2 \otimes f_1\rangle &= \begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0\end{bmatrix}^T \\
|e_1 \otimes f_2\rangle &= \begin{bmatrix}0 & 0 & 1 & 0 & 0 & 0\end{bmatrix}^T \\
|e_2 \otimes f_2\rangle &= \begin{bmatrix}0 & 0 & 0 & 1 & 0 & 0\end{bmatrix}^T \\
|e_1 \otimes f_3\rangle &= \begin{bmatrix}0 & 0 & 0 & 0 & 1 & 0\end{bmatrix}^T \\
|e_2 \otimes f_3\rangle &= \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 1\end{bmatrix}^T
\end{aligned}$$

In the qubit formalism one forsakes the cumbersome tensor product operation notation and simply writes $|0 \otimes 0\rangle = |00\rangle$, $|0 \otimes 1\rangle = |01\rangle$, so on. Note that generally we have $|000\rangle = |0\rangle \otimes |0\rangle \otimes |0\rangle$ (for the tensor product is associative) and the partition rule outlined above is easily generalizable to n-fold tensor products of Hilbert spaces $\bigotimes_i^n \mathbb{H}^i$. Notably, for the 2^n dimensional n-fold tensor product of two-dimensional Hilbert spaces $\mathbb{H}^i$, one may specify a basis vector

$$\bigotimes_i^n |e_i\rangle$$

(where each e_i is one of 1 or 0) which has a column vector representations with all zeroes and one unity at the entry

$$1 + \sum_{i=1}^{n} e_i \cdot 2^{i-1}. \tag{1}$$

As an example, consider $|110\rangle$. This has a column vector representation with unity in the entry

$$1 + 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 = 4$$

and thus

$$|110\rangle = \begin{bmatrix}0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\end{bmatrix}^T$$

In general, the matrix form of a tensor product of qubits can be written as a **Kronecker product**[1]; e.g.,

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix} \otimes \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{bmatrix} = \begin{bmatrix} v_1 w_1 \\ v_1 w_2 \\ \vdots \\ v_1 w_M \\ v_N w_1 \\ v_N w_2 \\ \vdots \\ v_N w_M \end{bmatrix}. \tag{2}$$

This is a particularly useful form of the tensor product because computations are primarily made with states cast in their matrix representation. We will also later see that certain operators as well as tensor products between kets and operators can be written as Kronecker products.

1.2 Entanglement

The preceding section saw an informal definition of the underlying mathematics of quantum computing and, in particular, composite qubit states. Here we motivate these definitions by ascribing a physical interpretation to Hilbert space and state ket tensor products.

A typical treatment of such objects appeals to the classical notion of *angular momentum addition*: angular momenta in the classical picture add vectorially, e.g., the total angular momentum is $\mathbf{J} = \mathbf{J}_1 + \mathbf{J_2}$.

In translating to quantum mechanics we would like to preserve this feature; in particular, we want the total angular momentum operator to yield, as an observable, the sum of the angular momenta of each rotating subsystem (themselves in turn observables of corresponding operators). Naively defining operators $\mathbf{J} = \mathbf{J}_1 + \mathbf{J_2}$, however, does not yield the appropriate model. Say $\mathbf{J}_1$ is the orbital angular momentum and $\mathbf{J}_2$ is the spin of a particle; a state with defined orbital angular momentum must then also have defined spin angular momentum, which is demonstrably untrue.

The appropriate model handles this natural separation of orbital and spin angular momentum by effectively subdividing the Hilbert space into two independent spaces: one reserved for orbit and another for spin. This is the significance of the tensor product

$$\mathbb{H}^A \otimes \mathbb{H}^B.$$

This way one can act on either angular momentum state while preserving the integrity of the other, *and* measure the total angular momentum by defining

$$\mathbf{J} = \mathbf{J}_A \otimes I + I \otimes \mathbf{J}_B$$

where the identity operator I preserves either Hilbert space A or B.

This same principle can be applied to the case of a two-particle system. We posit that one can (by classical analogy) vectorially sum the spins of either particle to obtain the spin of the two-particle system

$$\mathbf{S} = \mathbf{S}_A \otimes I + I \otimes \mathbf{S}_B$$

One can then write the state ket of a two-particle spin system by $|\pm\pm\rangle$, (analogously to $|01\rangle$, $|00\rangle$, etc.) where acting on such a state with $\mathbf{S}$ simply produces the sum of the spins of the states captured by the ket.

Spin is one phenomenon that can constitute a composite state ket; the quantum information theory formalism typically does not adhere to any one phenomenon, but it is useful to keep a mental picture of spin as being the observable underlying quantum circuits. This is particularly true when we consider the surprising effects of quantum entanglement. Two particles are entangled when one cannot write the state ket as a tensor product of two kets. For example, the state $|0\rangle \otimes |1\rangle = |01\rangle$ is not entangled, for we have a definite picture of the entire system—we know that the first particle is in a spin up state and the second is in a spin down state (or vice versa, depending on whether 1 corresponds to up or down states). Likewise, the state

$$\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

is not a tensor product of two computational basis kets but it *can* be written as a tensor product of two kets. Here we clearly see the motive behind this entanglement condition—upon expanding the tensor product we see that observing the first particle in say state 0 in fact gives us no information about the second particle, for $|01\rangle$ and $|00\rangle$ are both possible outcomes.

However, the state

$$\left|\Psi^+\right\rangle \equiv \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$$

cannot be written as a product of computational basis kets. The physical interpretation is that observing one of the particles—say the first particle—instantly gives us information about the second particle, for the second particle *must* be in the up state if the first is in the down state and vice-versa (hence Bell's inequality, Einstein's famous proclamation of quantum mechanics' "spookiness", so on). As a matter of fact, the above state comprises one of four states known as the **Bell basis**—a "maximally entangled[1]" basis.[2] The four

[1]Maximally entangled states have $\mathrm{Tr}\left(\rho_{A,B}^2\right) = 1/2$ where ρ_A is the reduced density matrix with respect to system A and ρ_B is the reduced density matrix with respect to system B.[1]

kets in the basis are:

$$\begin{aligned}\left|\Psi^{+}\right\rangle &= \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)\\ \left|\Psi^{-}\right\rangle &= \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)\\ \left|\Phi^{+}\right\rangle &= \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)\\ \left|\Phi^{+}\right\rangle &= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)\end{aligned}$$

One can in fact determine whether a state is entangled (i.e., whether a state can be factored as a tensor product of kets) using the **reduced density matrix**. The density matrix of a state $|\alpha\rangle$ is, of course,

$$\rho_{ij} = \langle i|\alpha\rangle \langle\alpha|j\rangle$$

The reduced density matrix is the *partial trace* of the density matrix, which is defined with respect some subsystem of the state (in the case of a binary tensor product space, this is one of two particles A and B); in braket notation (for say A):

$$\rho_A = \mathrm{Tr}_A(\rho) = \sum_i (I \otimes \langle i|)\, |\alpha\rangle\langle\alpha|\, (I \otimes |i\rangle)$$

where $|i\rangle$ are the basis kets for the Hilbert space of particle B. The operators on either side of $|\alpha\rangle\langle\alpha|$ effectively take the trace of particle B's contribution to the density matrix *only*. We can in fact write the terms of $\sum_i I \otimes \langle i|$ in matrix form using the Kronecker product as specified in (2):

$$I \otimes |0\rangle \doteq \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \otimes \begin{bmatrix}1\\ 0\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 0\\ 0 & 1\\ 0 & 0\end{bmatrix} \text{ and } I \otimes \langle 0| \doteq \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \otimes \begin{bmatrix}1 & 0\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix} \tag{3}$$

or for ρ_B:

$$|0\rangle \otimes I \doteq \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \otimes \begin{bmatrix}1\\ 0\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\\ 0 & 0\\ 0 & 0\end{bmatrix} \text{ and } \langle 0| \otimes I \doteq \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \otimes \begin{bmatrix}1 & 0\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\end{bmatrix} \tag{4}$$

It can be shown (with the derivation omitted here for brevity) that if the trace of ρ_A^2 or ρ_B^2 is less than unity one has an entangled state; we will use this in demos to verify that certain gates act on qubits as expected (see Baaquie (2013) pages 93-113).[1]

1.3 Selected Quantum Gates

Quantum gates are operators on some qubit space. For simplicity of computation they are often cast in matrix representation form in the computational basis. We focus our discussion on two gates with interesting quantum properties, but briefly note that the Hadamard and $\pi/8$ gates can in fact be used to construct new gates.[2]

- **Hadamard gate**:

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

The Hadamard gate is notable because it creates a superposition out of computational basis kets. For example:

$$H\,|0\rangle \doteq \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1 & 0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 1\end{bmatrix} \doteq \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

- **CNOT Gate**:

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

The CNOT gate is undestood to entangle a "target" qubit and a "controller" qubit; see Barnett (2009) pg. 247.[2] The action of the CNOT gate on the target qubit depends on the state of the controller qubit: If the control is $|0\rangle$ the target is unchanged; if the control is $|1\rangle$, the CNOT gate acts on the target as if it is a NOT (Pauli-X, or simply the Pauli matrix σ_x) gate.

It is not difficult to see how the CNOT gate creates entangled states. Loosely speaking, whether the target qubit was "flipped" depends on the state of the controller qubit. Thus providing a superposed controller qubit and subsequently collapsing said qubit leads to a collapse of the target qubit; the value of this collapsed target qubit depends on observed value of the controller qubit.

It is worth noting that one can create a Bell basis ket $|\Phi^+\rangle$ by operating on the control qubit with a Hadamard gate prior to applying the CNOT gate.

2 Implementation

2.1 Design

The `Quantum_Computing` module comprises four classes—each corresponding to the mathematical definitions outlined in Section 1.1—a handful of functions, and implementations of selected quantum gates as instances of the `NGate` class. The module was implemented with a hybrid functional and object oriented approach: the core mathematical objects are implemented as Python classes and auxiliary functionalities are implemented as Python functions. The class structure is as follows:

- `QubitState`: The `QubitState` class implements the quantum computational qubit state in an arbitrary composition of qubit spaces. That is, the qubit state is a 2^n-dimensional state ket in an n-fold composition of qubit spaces. A general qubit state class is defined (as opposed to, say, a composite qubit state and a child qubit class or vice versa) for two reasons: there is no functionality required of qubits that is not also required of composite qubits and vice versa, and the module's design centers qubit spaces as the generators of qubit states, thus eliminating the need to specify whether a qubit is in a composite qubit space.

  The `QubitState` constructor takes two arguments: the `CompositeQubitSpace` instance representing the qubit space this qubit state is in and the matrix representation of this qubit. The constructor ensures that the matrix representation has the appropriate dimensionality for the given qubit space, and throws a `MatrixRepresentationError` if this not the case.

  For syntactic sugar, the `QubitState` class implements the binary arithmetic operations `*`, `+`, and `-`. The multiplication operation calls the `tensor_product` method, which itself returns a new `QubitState` object represented by the Kronecker product of the two multiplicands' matrices; see (2). The addition and subtraction operators, similarly, sum and subtract the matrix representations of the qubit states and return a corresponding new `QubitState`.

  For convenience, the `QubitState` class is equipped with a `get_density_matrix` method as well as a functionally identical `density_matrix` property. This returns a `Density_Matrix` operator as defined in `operators.py`, the matrix representation of which is computed with respect to the object's qubit space computational basis. The method `QubitState.get_reduced_density_matrix` computes the parcial trace of this density matrix with respect to a state given by an index 0 or 1 using equations (3) and (4) (note that there is no support for reduced density matrices of states in larger composite qubit spaces; see §2.2).

- `CompositeQubitSpace`: The `CompositeQubitState` class implements a n-fold tensor product of qubit spaces. This is the most general implementation of qubit spaces (i.e., it is the parent class of the two-dimensional non-composite `QubitSpace`). The class hierarchy is designed in this fashion because there are functionalities and properties of composite qubit spaces that become special cases in the `QubitSpace` class (and not vice versa); separation of concerns inspires us to write one implementation of the most general form of these functionalities.

  The `CompositeQubitSpace` constructor takes as a required argument `n` the number of qubit spaces out of which this instance is composed. Being the only identifying property of a qubit space instance, the equality operation `__eq__` for two `CompositeQubitSpace` instances is overriden to check whether `self.n == other.n`.

  The central functionality of `CompositeQubitSpace` instances is obtaining the computational basis for this qubit space composition. The method `basis_from_indices` returns, as a `ndarray`, the matrix representation of the the computational basis ket specified by the given list of indices; i.e., if given `[1, 0, 1]`, the matrix representatin of the ket $|101\rangle$. The method `basis_ket_from_indices` is identical but wraps the matrix representation in a `QubitState` object. Both methods utilize (1) to compute the entries of the basis ket. Lastlty, `onb_matrices` uses a binary recursion to generate all possible combinations of n zeros and ones, calls `basis_from_indices` on each, then returns these `ndarray` matrix representations in a list, effectively producing a list of spanning orthonormal basis kets for this composite qubit space.

- `QubitSpace` (inherits `CompositeQubitSpace`): Motivated by the fact that a qubit space can appropriately be thought of as a 1-fold composite qubit space, the `QubitSpace` class is a child class of `CompositeQubitSpace` and thus inherits all functionality specified above.

  One feature of two-dimensional qubit spaces that is not possessed by composite qubit spaces is the ability to capture all information about a state ket using a Bloch sphere representation. The `QubitSpace` implementation leverages this picture in the `make_state` method, which produces `QubitState` instanaces and takes in either `alpha` and `beta` keyword arguments representing the azimuthal and polar angles, or a `coeffs` keyword argument representing the coefficients of the superposition of basis kets $|0\rangle$ and $|1\rangle$ (which the method normalizes). If given all keyword arguments, the method prioritizes angles; if one angle is missing, the method uses the coefficients.

- `NGate` (inherits `Operator`): the `NGate` class implements quantum n-gates—i.e., unary gates (2-by-2 matrices), binary gates (4-by-4 matrices), and virtually any other `Operator` on the appropriate qubit space. The class inherits the `Operator` class as gates can be regarded as quantum operators; its constructor takes in an array `x` as the matrix representation of this operator and the `CompositeQubitSpace` on which this operator acts, ensures that the given matrix has the appropriate dimensions for this qubit space and verifies that the matrix is unitary, throwing `MatrixRepresentationError` in the case either of these requirements is not fulfilled.

  `NGate` implements `__call__`, meaning that instances of this class are callable. Calling a `NGate` instance on a `QubitState` is equivalent to calling the method `NGate.apply` on that `QubitState`, which is also publicly accessible. The `apply` method multiplies the provided `QubitState`'s matrix representation by the instance's own matrix representation, and returns the resulting matrix wrapped in a `QubitState`.

2.2 Known Bugs and Limitations

- The current reduced density matrix implementaton does not support qubit states composed of more than one qubit. It is posited by the author that the partial trace is well-defined for larger compositions of qubit states. It remains to determine an algorithm to do so and implement it.

- `CompositeQubitSpace.onb_matrices` is perhaps functionally equivalent to simply generating the columns of the 2^n-by-2^n identity matrix; further investigation needed.

- Fuzz-testing revealed certain usages of analytical functions in `QubitSpace.make_state` gave rise to Numpy underflow errors. It is assumed that given values will not be sufficiently small to cause unexpected behavior; if they are, appropriate unit conversions must be made.
- The present test suite for the quantum computing module (`tests/Test_Quantum_Computing.py`) is quite limited due to time constraints; in particular, `QubitState.get_density_matrix`, `.get_reduced_density_matrix`, `.__add__`, `.__sub__`, `tensor_product`, and `NGate` have not been tested. It remains to write a more thorough test suite and produce a follow-up report including a summary of said test suite.

2.3 Refactoring

The previous PULSEE code-base has undergone some refactoring in preparation for the distribution of the software as a Python package. Source code has been relocated to the directory `src/pulsee`, following the "src" layout cited by Pytest's integration practices. File (module) names have been made all-lowercase per compliance with the Python style guide. The author also intends to, in the future, change class names to comply with the "CapWords" convention, also per Python's style guide.

2.4 Installation and Demo

A `setup.py` file has been created so that this development code can be installed, tested, and demoed (see the below demo for an example of importing and using `pulsee`). One can install a development version of the package to their Python environment by navigating to the directory containing `setup.py` and running `pip install -e .`; this development installation will update the environment's version of `pulsee` whenever any change to source files are made.

A demo Jupyter notebook of PULSEE quantum computing functionality can be found here. The GitHub repository containing the project files can be found here.

3 References

[1] Belal E. Baaquie. *Density Matrix: Entangled States*, pages 93–113. Springer New York, New York, NY, 2013.

[2] Stephen M. Barnett. *Quantum Information*. Oxford University Press, New York, NY, 2009.

[3] Wolfgang Scherer. *Mathematics of Quantum Computing*. Springer-Verlag, Cham, Switzerland, 2016.