QCLAB: A MATLAB Toolbox for Quantum Numerical Linear Algebra

Sophia Keip, Daan Camps, Roel Van Beeumen

Abstract

Quantum numerical linear algebra is about solving numerical linear algebra problems on quantum computers - a field that has seen exciting and significant progress in the past few years. Rapid advancements in quantum hardware continue to drive this momentum forward and highlight the fast-paced progress of the field. To facilitate quantum algorithm research, especially as quantum hardware is still maturing, access to robust computational tools is crucial. We present QCLAB (`https://github.com/QuantumComputingLab/qclab`) [1], an object-oriented MATLAB toolbox for creating, representing and simulating quantum circuits. What sets QCLAB apart is its emphasis on numerical linear algebra, prioritizing numerical stability, efficiency, and performance. This dedication to robust numerical techniques underlies its role as the foundational framework for a range of derived software packages and quantum compilers.

In this talk, featuring a MATLAB tutorial on QCLAB, we will not only showcasing the key features of QCLAB, but also providing an overview of the state of the art in quantum numerical linear algebra research. To offer concrete insights, the presentation will focus on three landmark quantum algorithms: the Quantum Fourier Transform (QFT) [2], Quantum Phase Estimation (QPE) [4], and the Quantum Singular Value Estimation (QSVE) [3]. By demonstrating QCLAB along the way and introducing fundamental concepts in quantum computing, the audience will be encouraged to engage actively with this promising research area.

The QFT is a quantum version of the discrete Fourier transform forming the foundation for numerous quantum algorithms in quantum numerical linear algebra. Building on the QFT, QPE is a principal quantum algorithm used to determine the eigenvalue (or phase) corresponding to an eigenvector of a unitary operator. In simple terms, it estimates the phase θ in the equation:

$$U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle,$$

where U is a unitary operator and $|\psi\rangle$ is an eigenvector of U. QPE is essential for applications like factoring large numbers (as in Shor's algorithm) and finding eigenvalues in quantum simulations. Finally, the QSVE extends these principles to non-unitary matrices, allowing singular values to be estimated directly through quantum methods. Both, QPE and QSVE promise polynomial complexity in n when applied to matrices of size 2^n.

To better understand how these quantum algorithms function, we will begin with the underlying principles of quantum computation, highlighting its accessibility for researchers with a linear algebra background [5, 6]. A quantum computation involves the following three key components:

- **Quantum State:** The representation of information, a unit vector in a complex vector space.
- **State Evolution:** The transformation of the quantum state via unitary operators.
- **Measurement:** The process of extracting information from the quantum state.

A common way of representing those three components is a so-called *quantum circuit*. A Quantum circuit is an intuitive visual way to track the evolution and measurement of a quantum state. QCLAB, based on this circuit model, offers a user-friendly interface for constructing, simulating, and visualizing quantum circuits, in line with most modern quantum hardware platforms.

Quantum states and qubits: A quantum circuit acts on a register of *qubits*, which hold quantum information. Qubits are the quantum counterpart to classical bits. While a classical bit is either 0 or 1, a qubit can exist in a linear combination, called *superposition*, of two *basis states* $|0\rangle$ and $|1\rangle$. To represent a state $|\phi\rangle$ of a single qubit as vector in $\mathbb{C}^2$, we choose the standard basis $|0\rangle = [1,0]^T$, $|1\rangle = [0,1]^T$. This leads to

$$|\phi\rangle = \alpha|0\rangle + \beta|1\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix},$$

where α and β are complex numbers, called *amplitudes*, and $|\alpha|^2 + |\beta|^2 = 1$. For multiple qubits, the state space grows exponentially. Consequently, the state of a n-qubit register corresponds to a vector in $\mathbb{C}^{2^n}$ and is described by a linear combination of 2^n basis states. These basis states are formed as tensor products of the 1-qubit basis states, i.e., $|b_1 b_2 \cdots b_n\rangle = |b_1\rangle \otimes |b_2\rangle \otimes \cdots \otimes |b_n\rangle$ with $b_i \in \{0,1\}$. Note that $b_1 b_2 \cdots b_n$ are bit strings that can be interpreted as the binary representation of the integers from 0 to $2^n - 1$. The n-qubit state can thus be represented as $\sum_{b=0}^{2^n-1} \alpha_b |b\rangle$, with normalization $\sum_{b=0}^{2^n-1} |\alpha_b|^2 = 1$. For instance, a 2-qubit register has the four basis states $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$ with $|00\rangle = |0\rangle \otimes |0\rangle = [1,0,0,0]^T$ and analogous expressions for the rest.

To set up a 2-qubit quantum circuit in QCLAB, we use the following code:

```
>> circuit = qclab.QCircuit(2);
```

q_0 —
q_1 —

where the circuit diagram on the right represents the empty qubits q_0 and q_1.

State evolution: Once we have prepared an n-qubit register in a certain state $|\psi\rangle \in \mathbb{C}^{2^n}$, we can evolve it over time. To ensure that the quantum state remains normalized, this evolution is achieved by applying unitary transformations $|\psi\rangle \to |\psi'\rangle$, known as quantum gates, which are represented by unitary matrices $U \in \mathbb{C}^{2^n \times 2^n}$ and are depicted in circuit formulas as blocks:

$$|\psi'\rangle = U|\psi\rangle, \qquad\qquad |\psi\rangle \text{ —}\boxed{U}\text{— } |\psi'\rangle$$

In theory, any unitary U can serve as quantum gate. However, in practice, the implementation of these gates is constrained by the underlying hardware, which determines which gates can actually be realized. An important 1-qubit gate is the Hadamard gate H, which transforms the basis states into equal superpositions, i.e. a linear combination with equal amplitudes.

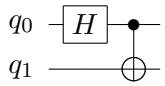
$$H|0\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right), \qquad\qquad H|1\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right).$$

A common 2-qubit gate is the controlled NOT gate, abbreviated to CNOT. This gate flips the state of the target qubit if and only the control qubit is in state $|1\rangle$. The unitary matrix representations of these two gates are

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \qquad\qquad \text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

QCLAB natively implements a wide variety of commonly used quantum gates as well as the option to implement your own gate based on its matrix representation. Going back to our example circuit, we can add a Hadamard gate to the first qubit (qubit 0) and a CNOT gate with control qubit 0 and target qubit 1 using the `push_back` function. On the right you see that our quantum circuit grows from left to right, which reflects the order the gates are applied to the state.

```
>> circuit.push_back(qclab.qgates.Hadamard(0));
>> circuit.push_back(qclab.qgates.CNOT(0,1));
```



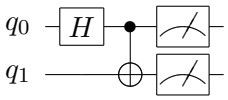
Measurement: In contrast to classical bits, it is not possible to observe the quantum state directly. Instead, we can only gain information through measurements, which inherently affect the state itself since they collapse the state to one of the basis states. In a 1-qubit state $\alpha|0\rangle + \beta|1\rangle$, the probability of measuring 0 is $|\alpha|^2$, while the probability of measuring 1 is $|\beta|^2$. After measurement, the state collapses to either $|0\rangle$ or $|1\rangle$, based on the observed outcome. For a general state $|\psi\rangle = \sum_b \alpha_b|b\rangle$, the probability of measuring a basis state $|b\rangle$ is

$$P(b) = |\alpha_b|^2.$$

Since $|\psi\rangle$ is normalized, this defines a valid probability distribution over the possible measurement outcomes. For instance, for a 2-qubit register in the state $\alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$, the probability of measuring 0 for both qubits is $|\alpha_{00}|^2$, and the system collapses to $|00\rangle$ after the measurement.

Let us add measurements to both qubits in our QCLAB example circuit:

```
>> circuit.push_back(qclab.Measurement(0));
>> circuit.push_back(qclab.Measurement(1));
```



Simulating quantum circuits: After constructing a circuit, the next step is to execute it and observe the results. Here, we set up a 2-qubit circuit consisting of a Hadamard gate on the first qubit, a CNOT gate with control qubit 0 and target qubit 1, and two measurements. Let us see what the circuit does on the initial state $|00\rangle = |0\rangle \otimes |0\rangle$. Applying the Hadamard gate to the first qubit yields

$$H|0\rangle \otimes |0\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \otimes |0\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle.$$

Next, the CNOT gate flips the bit of the second qubit whenever the first qubit is 1, so

$$\text{CNOT}\left(\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle\right) = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle.$$

Finally, by measuring both qubits, we measure $|00\rangle$ and $|11\rangle$ both with probability $|\frac{1}{\sqrt{2}}|^2 = 0.5$. To simulate the circuit in QCLAB, we use the `simulate` function with the initial state as input:

```
>> simulation = circuit.simulate('00');
```

For all measurements we can get the possible measurement results together with the corresponding probabilities and collapsed states by

```
>> simulation.results
    ans =  2x1 cell
        '00'
        '11'
```

```
>> simulation.probabilities
    ans =  2x1 cell
        0.5000
        0.5000
```

```
>> simulation.states
    ans =  2x1 cell
        [1;0;0;0]
        [0;0;0;1]
```

QCLAB provides a full state simulation, meaning it accurately represents the entire quantum state vector, allowing precise tracking of amplitudes and phase information for each qubit throughout the computation. The straightforward simulation of quantum circuits in QCLAB makes it a valuable

tool for rapid prototyping of quantum algorithms. This capability allows researchers to efficiently experiment with and refine their algorithms prior to moving on to more advanced stages.

Additional features: Other than the computational tasks, QCLAB enables the visualization of quantum circuits directly in the MATLAB command window and supports saving a circuit diagram to LaTeX source files, as demonstrated in the diagrams presented within this abstract. Both can be done using the following commands:

```
>> circuit.draw;
>> circuit.toTex;
```

This functionality makes it particularly useful for research documentation and presentations. QCLAB also provides input/output compatibility with openQASM, a low-level programming language used to describe quantum circuits, which is compatible with quantum hardware. This allows users to test their quantum circuits on real quantum computers and is achieved by the command:

```
>> circuit.toQASM;
```

Alongside the numerical linear algebra applications we present in this talk, QCLAB offers a variety of other examples that help users getting familiar with both quantum computing concepts and the toolbox itself. Additionally, extensive documentation is available to make the learning process as smooth as possible.

QCLAB also has a C++ counterpart, QCLAB++ [7, 8], which is designed for more computationally demanding tasks by leveraging GPU capabilities. QCLAB++ retains the same user-friendly syntax as QCLAB, allowing researchers to easily transition from prototyping in MATLAB to scaling up simulations with C++ on GPUs.

This talk is designed for both researchers in numerical linear algebra seeking an easy entry point into quantum computing, and experienced quantum computing practitioners looking for a tool to facilitate rapid prototyping of quantum algorithms.

References

[1] D. Camps and R. Van Beeumen. "QCLAB v0.1.2," Aug. 2021. doi:10.5281/zenodo.5160555. github.com/QuantumComputingLab/qclab

[2] D. Coppersmith. "An approximate Fourier transform useful in quantum factoring." IBM Research Report RC 19642, (1994).

[3] A. Gilyén, Y. Su, G. H. Low and N. Wiebe. "Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics." In Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (pp. 193-204), (2019).

[4] A. W. Harrow, A. Hassidim and S. Lloyd. "Quantum algorithm for linear systems of equations." Physical review letters, 103(15), (2009).

[5] G. Nannicini. "An introduction to quantum computing, without the physics." SIAM Review 62.4: 936-981, (2020).

[6] M. A. Nielsen and I. L. Chuang. "Quantum computation and quantum information," Vol. 2. Cambridge: Cambridge University Press, (2001).

[7] R. Van Beeumen and D. Camps. "QCLAB++ v0.1.2," Aug. 2021. doi:10.5281/zenodo.5160682. github.com/QuantumComputingLab/qclabpp

[8] R. Van Beeumen, D. Camps, and N. Mehta. "QCLAB++: Simulating Quantum Circuits on GPUs," arXiv:2303.00123 (2023).