

Simulation of Digital Quantum Circuits

Hiroki Kawai, Lukas Rosario, Dhyllan Skiba

May 2, 2021

1 Introduction

Quantum computing is the use of quantum information to perform computations. Given its quantum nature, quantum computing is a promising technology to simulate quantum problems which are hard to be simulated on the existing classical computers due to e.g. the sign problem with the Monte-Carlo simulations. For example, many quantum algorithms have been proposed for quantum many-body problems for condensed matter physics and quantum chemistry, as well as recently the applications are discussed for the gauge theories with a discrete gauge symmetry such as \mathbb{Z}_2 [2] and potentially, those with a continuous Lie group gauge symmetry e.g. QCD. In addition, since the invention of the Shor's algorithm [5] which can perform integer factorization in a polynomial time, quantum computing has attracted other fields than natural science, such as finance, IT security, and more. Even though quantum bits (qubits) with controllable entanglements have been realized with a variety of hardware technologies including superconductive materials and ion-trapping, the undesired interactions between the qubits and the external controlling devices lead to non-negligible noise in the computation processes and even worse, the complete decoherence of the qubits. In order to realize the fault-tolerant (FT) quantum computers, which are the necessity for the industrial applications, the devices must consist of more than thousands of physical qubits with very high precision to run quantum error correction codes, whereas any of the existing devices have no more than a hundred physical qubits. In other words, we are in the very early stage of the noisy intermediate-scale quantum (NISQ) era [4]. Studying the NISQ devices is still very important to estimate the potential of the future FT quantum devices, so we need to simulate them on a classical computer to evaluate the performance of those noisy devices. In this work, we implement a simple classical simulator of quantum circuits, which can perform any one-qubit unitary gates and the CNOT gate. It is hard to efficiently run the simulations of a universal digital quantum circuit on classical computers due to the exponential nature of the number of the dimension of the Hilbert space in which quantum information lives. We use the sparse matrix representations to represent the wavefunctions so that one can alleviate this exponential cost. Nevertheless, let us also note that one can efficiently simulate a quantum circuit on a classical computer if the circuit has only Clifford gates by sacrificing the universality of the quantum circuit known as the Gottesman–Knill theorem [1], or even for the simulations of the universal quantum circuits, one can reduce the exponentiality further by using the matrix product state (MPS) [6] which explicitly takes the entanglements between the qubits into account to reduce the total degrees of freedom. Our simulation is tested with two simple testbenches of the quantum Fourier transform and the time evolution of the transverse-field Ising model.

2 Mathematical Background

The basic information unit of quantum computers is called a qubit, which is a 2-state quantum system or a spin-1/2 system. We usually denote the up spin as the $|0\rangle$ state and the down spin as the $|1\rangle$ state to make the comparison to a classical bit clearer. More mathematically saying, a single-qubit state can be represented as a two dimensional vector living in the two dimensional complex Hilbert space, and a quantum gate acting on the single qubit is a fundamental **2** representation of the $SU(2)$ group acting on the Hilbert space, i.e. a time evolution with the unitary rotation $U = \exp(itH) \in SU(2)$ under the Hamiltonian $H \in \mathfrak{su}(2)$ described by the Schrödinger equation. Since the $SU(2)$ group has three generators σ^j :

$$\sigma^1(=X) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma^2(=Y) = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma^3(=Z) = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad (2.1)$$

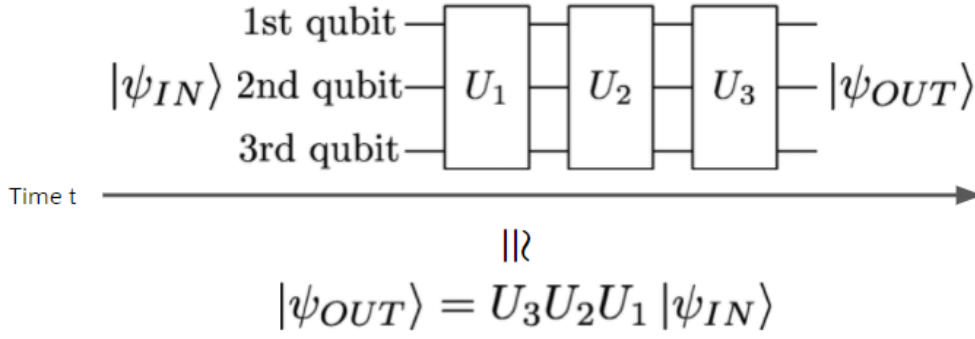


Figure 1: A schematic diagram of a quantum circuit and its correspondence to its matrix representation. $|\psi_{IN}\rangle, |\psi_{OUT}\rangle \in \mathcal{H}$ are the 2^N -dimensional complex vectors and U_i are the $2^N \times 2^N$ -dimensional complex matrices.

any $SU(2)$ rotation can be parametrized with three angles $\theta_j \in [0, 2\pi)$ with $j = 1, 2, 3$ as

$$U(\theta_1, \theta_2, \theta_3) = \exp \left(i \sum_{j=1}^3 \theta_j \sigma^j \right) \quad (2.2)$$

When we have $N > 1$ qubits, the entire Hilbert space becomes

$$\mathcal{H} = \bigotimes_{i=1}^N \mathcal{H}_i \quad (2.3)$$

where \mathcal{H}_i is the two dimensional Hilbert space corresponding to the i -th qubit, and hence the total dimension of \mathcal{H} is 2^N . An ideal quantum computer can perform any $SU(2^N)$ rotations acting on \mathcal{H} , whose generators are

$$\{I_{2 \times 2}, \sigma^1 \sigma^2, \sigma^3\}^{\otimes N} \setminus \{I_{2^N \times 2^N}\} \quad (2.4)$$

where $I_{n \times n}$ is the $n \times n$ dimensional identity matrix. Actually, the subtraction of $\{I_{2^N \times 2^N}\}$ is trivial since it generates only the trivial phase $U(1)$, and the generated group becomes $U(1) \times SU(2^N) = U(2^N)$ if one contains this trivial identity to the generator set. It is proven that any $(S)U(2^N)$ rotations can be realized with CNOT gates and one-qubit $(S)U(2)$ rotations [3], which our simulator can simulate. As a matrix acting on the entire \mathcal{H} space, any one qubit gate G in the 2×2 representation acting on the i -th qubit can be represented with the $\bigotimes_{i=1}^N \mathbf{2}$ representation of $SU(2)$ as

$$\left(\bigotimes_{k=1}^{i-1} I_{2 \times 2} \right) \otimes G \otimes \left(\bigotimes_{k=i+1}^N I_{2 \times 2} \right) \quad (2.5)$$

On the other hand, the CNOT gate is not a group element of $SU(2)$ but of $SU(3)$ and larger $SU(N)$. This allows the quantum computer to extend its computational ability to $SU(2^N)$. The sparsity of the full $2^N \times 2^N$ -dimensional representations of the one and two-qubit gates are promised by their locality, or the reducibility of the representations. In particular, the full representations of the one-qubit gates have no more than 2^{N+1} non-zero elements, and that of the CNOT gate has 2^N ones, which are both significantly less than the full 2^{2N} elements. A process of quantum computation can be represented as the matrix multiplications of the $2^N \times 2^N$ -dimensional gate matrices on the input qubit state vector, and it is usually schematized as Fig. 1

3 Implementation

For this project, we are implementing useful one-qubit gates and a single two-qubit gate, CNOT. The list of the built-in gates is Table 1. We compute the full $2^N \times 2^N$ -dimensional matrix representations of the one-qubit gates as (2.5) where

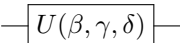
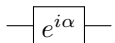

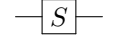
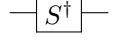
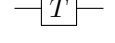
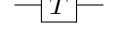
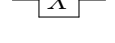
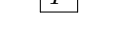

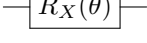
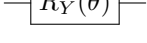
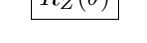

Gate name	Diagram	Matrix representation
General $SU(2)$		$\begin{pmatrix} \cos(\beta/2) & -e^{i\delta} \sin(\beta/2) \\ e^{i\gamma} \sin(\beta/2) & e^{i(\delta+\gamma)} \cos(\beta/2) \end{pmatrix} = R_Z(\beta)R_Y(\gamma)R_Z(\delta)$
Trivial $U(1)$ phase		$e^{i\alpha}$
Hadamard		$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
S		$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
S^\dagger		$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
T		$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$
T^\dagger		$\begin{pmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{pmatrix}$
Pauli- X		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli- Y		$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli- Z		$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Rotation around X -axis		$\begin{pmatrix} \cos(\theta/2) & -i \sin(\theta/2) \\ -i \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$
Rotation around Y -axis		$\begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$
Rotation around Z -axis		$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$
CNOT		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

Table 1: The built-in quantum circuit components for our simulator.

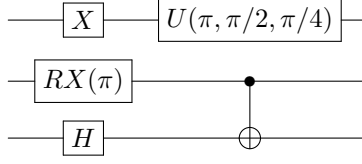


Figure 2: The example circuit.

the tensor product \otimes in the matrix representation is defined as

$$\otimes : (\mathbb{C}^{m \times n}, \mathbb{C}^{M \times N}) \rightarrow \mathbb{C}^{mM \times nN}, \quad A \otimes B = C \text{ where } C_{im+k, jn+l} = A_{i,j} B_{k,l} \quad (3.1)$$

The matrix representation of the CNOT gate is computed directly as its left regular representation acting on the qubit states in the computational basis. Let us again note these one and two-qubit quantum gates are very sparse. As the number of qubits in the system grows, the quantum gates get exponentially larger, thus it is essential that we take advantage of the sparse nature of these matrices. This not only decreases the amount of storage necessary to store the gate matrices, but it also decreases the number of operations executed during matrix-matrix and matrix-vector operations. We decided to implement all of our vectors and matrices in compressed sparse row matrix format (CSR). To do so, we made use of the Eigen C++ library. Eigen provides a nicely wrapped format for constructing and operating on sparse matrices. The Eigen library, however, does not come with tensor product functionality, so we implemented the function ourselves using Eigen sparse matrix formatting. To construct the quantum circuit, we developed a simple frontend interface in which you can add circuits by name and input parameters into the function. Here is some example code constructing a simple 3-qubit circuit which is equivalent to the circuit depicted in Fig. 2:

```
QCCircuit circ(3); // Initialize a 3-qubit circuit
circ.addX(0); // Add X gate acting on qubit-0
circ.addH(2); // Add H gate acting on qubit-2
circ.addRX(1, PI); // Add rotation X gate on qubit-1 with angle PI
circ.addU1(0, PI, PI/2.0, PI/4.0); // Add general SU(2) rotation
circ.addCNOT(1, 2); // Add CNOT with qubit-1 as control and qubit-2 as target
```

These `add...` functions add the information of the gates which are the gate name, qubit indices they are acting on, and the rotation angles, into the member list of the `QCCircuit` object. The `QCCircuit` object computes the explicit matrix representation of the circuit by calling its private `_compMatrix` member function when it needs it such as the `getUSparse` and `getUDense` member functions which return the sparse and dense matrix representations of the circuit respectively and inside of the `outputState` member function which receives the input state vector in the sparse matrix format as its argument, and compute the output state vector. The computation of the matrix is managed by the flag `isUComputed` so that the object computes the matrix representation only once and reuses it until the user adds other gates. The `QCCircuit` object has other functions that simulate the situation of the physical experiments, such as `expVal` which computes the expectation value measuring an one-qubit Pauli with the output state:

$$\langle \sigma_i^j \rangle = \langle \psi_{out} | \sigma_i^j | \psi_{out} \rangle = \langle \psi_{in} | U^\dagger \sigma_i^j U | \psi_{in} \rangle \quad (3.2)$$

and `ZBasisSampling` which simulates the distribution of the random finite sampling of the output state in the computational basis.

4 Results

4.1 Quantum Fourier transform

One simple but interesting test bench is the quantum Fourier transform (QFT), which is defined as

$$QFT |x\rangle = \frac{1}{\sqrt{2^N}} \sum_{k=0}^{2^N-1} e^{ikx} |k\rangle \quad (4.1)$$

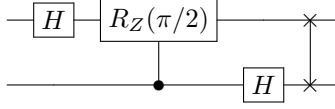


Figure 3: The circuit for the QFT with two qubits.

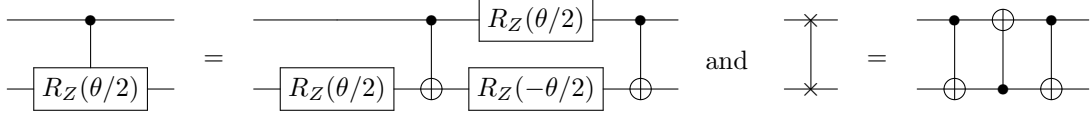


Figure 4: The lower level implementations of the controlled- R_Z gate and the SWAP gate.

where $|x\rangle$ is one of the computational-basis state. Due to its linearity, if the QFT takes a linear combination of the computational-basis states, then the output state is also a linear combination of the RHS of (4.1). The matrix representation of the QFT in the computational basis is

$$\mathcal{QFT}_{j,k} = \omega^{jk} \quad (4.2)$$

where $\omega = e^{i2\pi/2^N}$. Notice that ω is a generator of \mathbb{Z}_{2^N} , so $\omega^n = \omega^{n+2^N m} \forall n, m \in \mathbb{Z}$. For the $N = 2$ qubits case the matrix looks like

$$\mathcal{QFT}_{N=2} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & 1 & \omega^2 \\ 1 & \omega^3 & \omega^2 & \omega \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \quad (4.3)$$

with $\omega = e^{i\pi/2} = i$. The circuit diagram of $\mathcal{QFT}_{N=2}$ is Fig. 3, where the lower level gate structure of the controlled- R_Z and SWAP gates are as Fig. 4. The matrix representation of $\mathcal{QFT}_{N=2}$ computed by our simulator is as Fig. 5, which matches with the exact representation 4.3 up to the round-off error of the `exp` function in C++.

To analyze the performance of our simulator, we recorded the runtime of \mathcal{QFT}_N for N ranging from 2 to 10. As seen in Fig. 6, the runtimes are split into two sections. One for the computation of matrix representation of the \mathcal{QFT} , and the other for the computation of the output state given an input of $|0\rangle$. The reason we separate these is that the matrix representation of the \mathcal{QFT} only has to be calculated once, and then we can input any number of Hilbert spaces of the corresponding dimension at a much faster speed. Note that both of the runtime components grow at an exponential rate. This is due to the fact on a classical computer, the state space of N qubits is represented by a 2^N -dimensional vector. At 10 qubits, for example, the matrix representation of the quantum circuit is of dimensions 1024×1024 , and the Hilbert state space is also of dimension 1024.

Figure 6 also shows that the procedure of generating the matrix representation of the \mathcal{QFT} is quite computationally expensive in comparison to just calculating an output state. This makes sense because the matrix generation consists of increasing amounts of matrix-matrix multiplications and tensor products, whereas the output calculation is a singular matrix-vector multiplication.

4.2 Simulation of the transverse-field Ising model (TFIM)

For another test case of our simulator, we simulate the real-time evolution of the transverse-field Ising model with four spin-1/2 sites in one dimension with the periodic boundary conditions. The Hamiltonian of the model is defined as

$$\hat{H}_{TFIM} = -J \sum_{\langle x,y \rangle} \sigma_x^3 \sigma_y^3 - g \sum_x \sigma_x^1 \quad (4.4)$$

in $\mathfrak{su}(2)$ where $\langle x,y \rangle$ specifies a combination of neighborhood spins. Since the neighborhood interaction terms and the magnetic field term do not commute with each other, we use the Trotter-Suzuki approximation to simulate the time evolution operator $e^{-i\hat{H}_{TFIM}t}$ on a quantum circuit. The Trotter-Suzuki formula based on the theorem stating that

$$e^{AB} = \lim_{n \rightarrow \infty} \left(e^{A/n} e^{B/n} \right)^n \text{ where } [A, B] \neq 0 \quad (4.5)$$

```

circuit matrix:
      (0.5,0)      (0.5,0)      (0.5,0)      (0.5,0)
      (0.5,0)      (7.85046e-17,0.5)      (-0.5,0)      (-7.85046e-17,-0.5)
      (0.5,0)      (-0.5,0)      (0.5,0)      (-0.5,0)
      (0.5,0)      (-7.85046e-17,-0.5)      (-0.5,0)      (7.85046e-17,0.5)

```

Figure 5: The matrix representation of $\mathcal{QFT}_{N=2}$ computed by our simulator. The first value in the bracket is the real part of the complex element, and the second is the imaginary part.

So we approximate the time evolution operator as

$$e^{-i\hat{H}_{TFIM}t} \approx \left(e^{iJ(t/n)\sum_{\langle x,y \rangle} \sigma_x^3 \sigma_y^3} e^{ig(t/n)\sum_x \sigma_x^1} \right)^n = \left(\prod_{\langle x,y \rangle} e^{iJ(t/n)\sigma_x^3 \sigma_y^3} \prod_x e^{ig(t/n)\sigma_x^1} \right)^n \quad (4.6)$$

This approximation can be interpreted as the discretization of the evolution time with the interval of t/n , and iterate this discretized time step n times to reach the approximated time evolution of time t . The $e^{ig(t/n)\sigma_x^1}$ rotation is just a one-qubit R_X rotation with angle of $-2g(t/n)$, whereas the other $e^{iJ(t/n)\sigma_x^3 \sigma_y^3}$ rotation is a two-qubit rotation which can be implemented in the lower level as Fig. 7. We simulate the expectation value

$$\langle \sigma_0^3 \rangle = \langle 0 | e^{i\hat{H}_{TFIM}t} \sigma_0^3 e^{-i\hat{H}_{TFIM}t} | 0 \rangle \quad (4.7)$$

with two different time intervals $t/n = 0.1, 0.5$, and the result is as Fig. 8 where the exact values are computed by Mathematica. The result demonstrates that our simulator correctly simulates the quantum circuit for the time evolution with a high precision with enough small time interval t/n .

5 Summary

We implement a digital quantum circuit simulator with the various one-qubit gates and the CNOT gate with C++ and the library for linear algebra called Eigen. The qubit states and the gate operations are represented as complex sparse matrices in the full 2^N -dimensional complex Hilbert space to speedup the computational time and reduce the memory cost, even though it is still not efficient due to the exponentiality. We test the simulator with two testbenches of the quantum Fourier transform and the time evolution of the qubit state under the transverse-field Ising model Hamiltonian, and both of the simulations demonstrate that our simulator works as expected. For a future work, we have to think of parallelizing the expensive parts of the linear algebra computations such as the matrix multiplications and the tensor products, and also comparing it with other simulators such as qiskit for its computational time and the resource cost.

References

- [1] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Phys. Rev. A*, 70:052328, Nov 2004.
- [2] Erik J. Gustafson and Henry Lamm. Toward quantum simulations of \mathbb{Z}_2 gauge theory without state preparation. *Phys. Rev. D*, 103(5):054507, 2021.
- [3] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.
- [4] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018.
- [5] Peter W. Shor. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Sci. Statist. Comput.*, 26:1484, 1997.
- [6] Guifré Vidal. Efficient classical simulation of slightly entangled quantum computations. *Phys. Rev. Lett.*, 91:147902, Oct 2003.

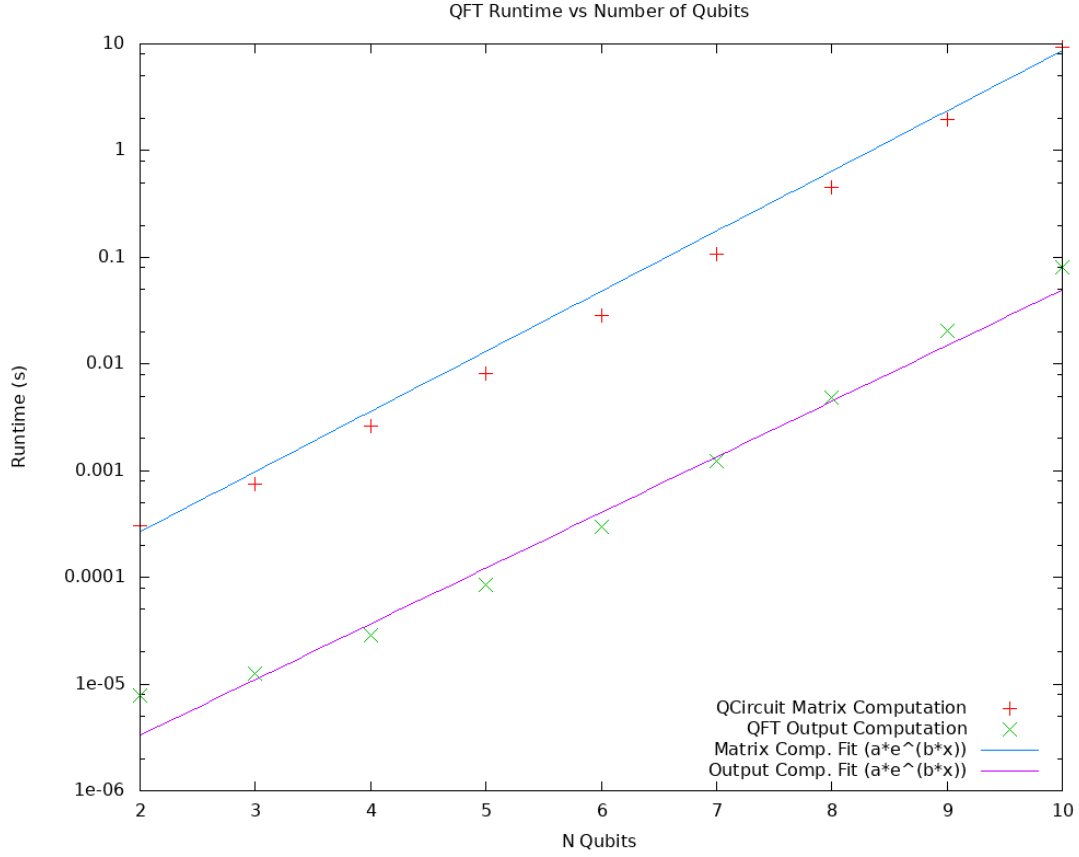


Figure 6: Runtime of our digital simulation of QFT_N for $N = 2$ to $N = 10$. The runtimes are split into two sections: the computation time for the matrix representation of the quantum circuit, and the computation time of the output of the QFT_N given an input of $|0\rangle \in \mathcal{H}^{2^N}$. The solid lines are exponential fits to both series.

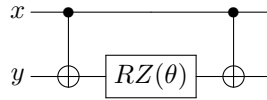


Figure 7: The lower level implementations of the two-qubit rotation $e^{i(\theta/2)\sigma_x^3\sigma_y^3}$.

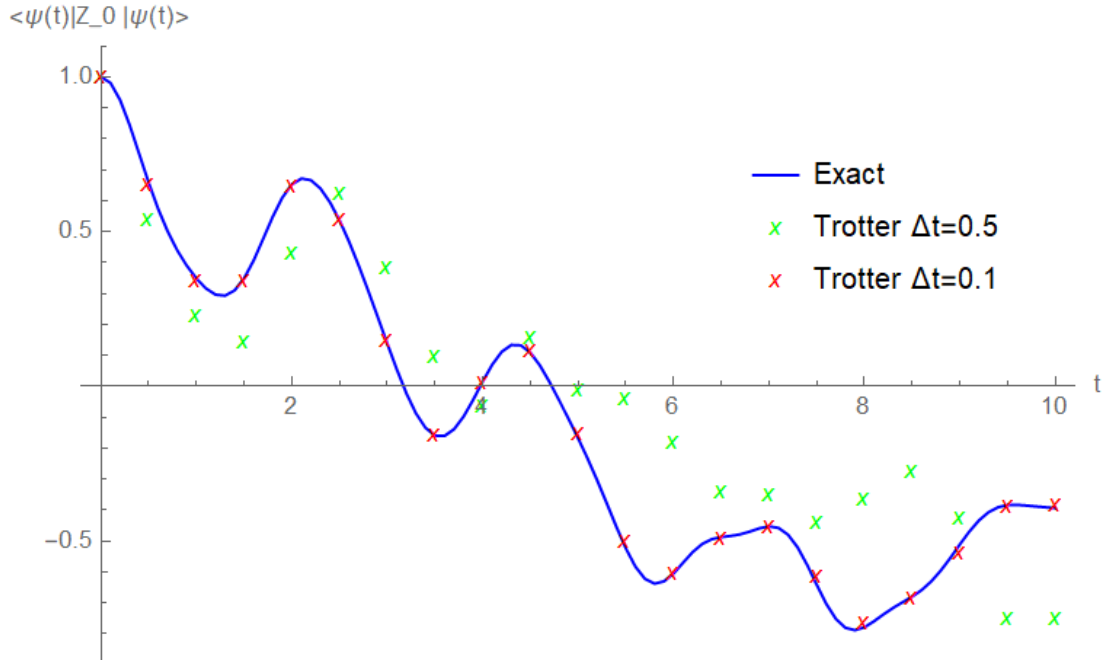


Figure 8: The simulation result of the time evolution of the expectation value of the σ_0^3 observable with the 4-qubit initial state $|0\rangle$ under the TFIM Hamiltonian.