Instruction Material

Simulating quantum many-body dynamics on a current digital quantum computer

Day 1

Practical Training
Condensed Matter Theory
Technical University Munich
Physics Department

Responsible: Prof. Dr. Michael Knap

Dates (Day 1/Day 2):
June 1/2, 2023

1 Preliminary for the Practical Training

Before you participate in the practical training, there are a few things that might get you prepared

1. Some minimal knowledge about quantum computation will be helpful, If you haven't learned any quantum computation before, don't worry. A comprehensive reference will be the first chapter of *Quantum Computation and Quantum Information* by Nielsen and Chuang.

2. It would be very helpful if you briefly read through the content for the next day in advance, especially when some new theoretical concepts are introduced. Please also feel free to prepare questions if there are any. During the practical, you will be completing the exercises with help from the instructors.

3. Before we start coding in Qiskit, we will give an introduction and some time for you to install Qiskit.

4. For the coding exercises, we advise you to use interactive Python notebooks. You can for example install your setup via Anaconda, Miniconda or create a Google account to work online with Google Colab.

5. The practical training consists of Day 1 (basics) and Day 2 (application), you are expected to hand in a written report for **both** Day 1 and Day 2, respectively. You can also choose to only participate on Day 1.

6. During the exercises, we will either **simulate** the circuits to get full access to the wavefunctions or determine quantities by **measuring/ sampling** circuits, which mimics the use of a real quantum computer. Pay attention to the highlighted keywords.

2 Quantum Computation with Superconducting Qubits

Quantum computers are platforms (highly controlled quantum systems) where computation can be performed following the principles of quantum mechanics. Unlike classical computers, quantum computers harness quantum mechanical properties such as superposition and entanglement. For this reason, quantum computers are expected to be more powerful than the classical computers.
Enormous progress has been made over the last decades in building a universal and practical quantum computer. Nowadays, many quantum platforms are available, such as cold atoms in optical lattices, superconducting qubits, and trapped ions, supporting roughly 10$\sim$100 qubits for universal computation. These quantum platforms are *Noisy Intermediate-Scale Quantum* (NISQ) computers. Namely, they are not fault-tolerant and suffer from noise due to interaction with the environment. Among these platforms, superconducting qubits are, for their flexibility

and scalability, one of the most popular candidates to build a large-scale quantum computer. In 2019, the milestone, so-called *quantum supremacy*, was claimed to be first demonstrated on a 53-qubit superconducting quantum processor (Sycamore) in Google [1].

In this practical training, you will learn about how to write codes that can be run directly on a superconducting quantum computer using Qiskit! Now you might wonder, **what is the superconducting quantum computer?** The superconducting quantum computer is based on an architecture called the *circuit quantum electrodynamics* (cQED). The superconducting qubits are some circuits consisting of inductors, capacitors, and Josephson junctions built from superconducting materials. The circuitry isolates two energy levels of the system protected by the superconducting energy gap. Within this architecture, the single-qubit gate is applied by coupling the qubit with some microwave pulses, and the two-qubit gates can be applied by resonating the frequency of the two qubits or activating the inter-qubit interaction by microwave sources. By choosing an optimized set of parameters for these operations, the superconducting qubits are equipped with a set of supported (universal) gates.

Compared to other quantum computers, superconducting qubits have the advantages of being scalable and flexible in terms of design and control. The fabrication of superconducting qubits is based on existing chip-making and semiconductor technologies, which are already rather mature. The design of each superconducting qubit can also be further improved and accommodate different needs by changing the underlying circuitry. Superconducting quantum computer supports fast gate operation (such as the Transmon qubits used in Sycamore) in tens of nanoseconds, whereas the trapped ions, for example (depending on the specific system) can take tens or hundreds of microseconds. This renders fast quantum computation. But superconducting qubits also come with a short coherence time (e.g. tens of microseconds for Transmon qubits, compared to the coherent time of seconds in an ion trap). This strongly limits the number of operations we can do on the superconducting qubits.

It is still an active area of research, and not yet clear, as to which types of quantum platforms might serve the best as a practical quantum computer. Superconducting qubits are surely one of the leading candidates. There is a whole branch of area of research about superconducting qubits and it is rapidly expanding. Interested students can refer to, e.g. [2, 3] for a survey of state-of-art superconducting qubits and links to other references.

3 Error-mitigation on NISQ Devices

The computation performed on the current, and possibly all the near-term quantum computers will be noisy, i.e. the results of the computation are subject to various types of errors, such as decoherence errors, gate errors, readout errors, etc. Therefore, error mitigation is an important part of quantum computing on NISQ devices. In general, understanding these errors is a very complicated task and is

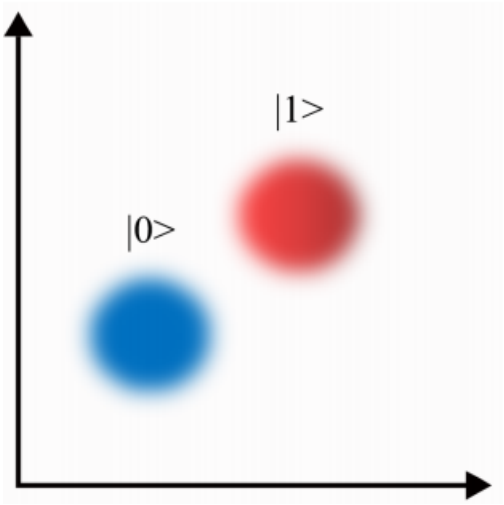

Figure 1: *Measured signals from some experiments. The state is identified by the clustering. If the two clusters have overlap, an incorrect readout can happen. Figure taken from [2].*

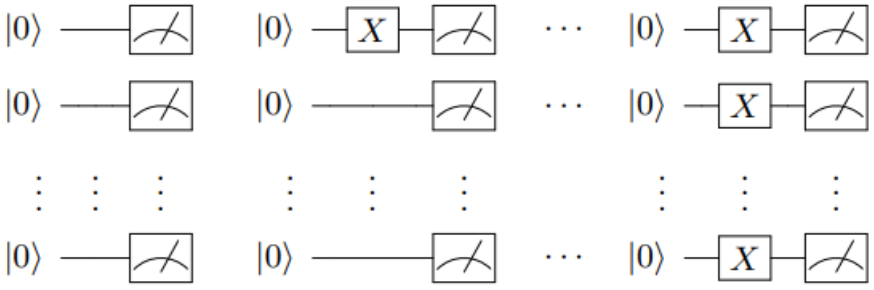

Figure 2: *The calibration circuits. Figure taken from [4]*

an area of intense research. The errors depend on the machine and architecture used, and many of them are a result of complicated quantum interaction.
On NISQ devices, we will not be able to perform full error correction. Apart from the hardware improvement, we can instead mitigate the errors by some effective error modeling, and post-processing the data based on some calibration experiments. In this practical, we will implement a simple, but effective protocol to mitigate the readout errors, called the readout matrix unfolding [4].
The readout errors is an important class of errors on NISQ device. They are the errors that occur when measuring the outcomes of a computation. For example, the measurement process itself can cause transitions between $|0\rangle$ and $|1\rangle$, either because it takes longer than the coherence time of the states, or some leakage of (e.g. in superconducting qubits) microwave pulses. It is also possible to simply read the wrong state if the cluster of the measured signals for the two states have overlap (Fig. 1).
The technique we will be using consists of two steps, the calibration and the

unfolding. The calibration is a series of experiments performed on the device before the calculation.

Calibration: Suppose there are N qubits in the devices. Starting from the product states $|00...0\rangle$ (which is usually the initial state on a quantum computer), we can prepare 2^N (computational) basis states. For each prepared basis state $|b\rangle$, we perform measurement on all the qubits (see Fig. 2). This is then repeated m times (which, e.g. can be the same as the number of measurements performed for readout the computation result). Therefore we perform in total $m2^N$ experiments. We can represent the data in a matrix form by

$$P_{ab} = \Pr\left(\text{Measured bitstring } a | \text{ Prepared bitstring } b\right) \tag{3.1}$$

we call this the *readout matrix* for the calibration, it contains the empirical probability distribution for each bitstring readout.

Unfolding: The unfolding step is to use P_{ab} to mitigate the errors in the measured probabilities for each string. Suppose we have the vector of measured probabilities for each bitstring v'_a, and the error-free probabilities are v_a. The assumption we make is they are related by (at least for small error rates)

$$v'_a = \sum_b P_{ab} v_b \tag{3.2}$$

which is just a matrix equation. Therefore, the simplest approach to get v_a from v'_a is doing a matrix inversion of P (this is basically what Qiskit does to mitigate readout errors, try it out yourself once you know more about Qiskit!).

> ***Exercise 1:*** Show that if the measured v'_a sums to unity $\sum_a v'_a = 1$, the inverted v_a preserves the sum, i.e. $\sum_a v_a = 1$.

However, the simple matrix inversion may fail for a couple of reasons. The main issue is the inverted v_a can become unphysical, it has negative entries despite that it should be a probability. The issue could be avoided by using more advanced unfolding techniques, such as solving the linear system Eq. (3.2) with constraints. Interested students can refer to [4] for a full discussion. We need to keep in mind that we should always check whether the inverted results are physical when we implement the method in practice.

4 Qiskit Basics

Qiskit is an open-source Python library developed by IBM for simulating and running quantum circuits. In this practical, you will get to learn how to code in Qiskit and use it to simulate interesting quantum many-body systems. Qiskit is also directly implementable on the real quantum computer provided by IBM (the codes are also shared by other platforms). So, you might get a chance to run your quantum many-body simulation on a real superconducting quantum computer!

We will introduce the Qiskit basics using this Python notebook. You could also install Qiskit on your laptop if you prefer. You can find detailed instructions in here.
Use what you have just learned to complete the following exercises (you can find even more textbook example problems solved using Qiskit on the documentation page here).

4.1 Quantum Teleportation

Quantum teleportation is a technique used to transmit (teleport) a quantum state from Alice to Bob, given they pre-share an entangled state. Let us consider teleporting a single-qubit state. Suppose Alice and Bob pre-share an EPR pair $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. Now Alice has the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ that she wants to teleport to Bob by classical communication. The system now has the wavefunction

$$\begin{aligned}
&\frac{1}{\sqrt{2}}|\psi\rangle_M(|0\rangle_A|0\rangle_B + |1\rangle_A|1\rangle_B) \\
&= |\Phi^+\rangle_{MA}(\alpha|0\rangle_B + \beta|1\rangle_B) + |\Phi^-\rangle_{MA}(\alpha|0\rangle_B - \beta|1\rangle_B) \\
&\quad + |\Psi^+\rangle_{MA}(\alpha|1\rangle_B + \beta|0\rangle_B) + |\Psi^-\rangle_{MA}(\alpha|1\rangle_B - \beta|0\rangle_B)
\end{aligned} \tag{4.1}$$

where M refers to the message qubit Alice holds. A, B refers to Alice and Bob. And we define the four EPR pairs

$$\begin{aligned}
|\Phi^+\rangle &= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \\
|\Phi^-\rangle &= \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \\
|\Psi^+\rangle &= \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \\
|\Psi^-\rangle &= \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)
\end{aligned} \tag{4.2}$$

They form a complete basis. If Alice performs a measurement on EPR pairs, Bob's qubit will collapse to one of the four states in (4.1). Alice then sends her measurement results encoded in two classical bits and Bob can recover the message state $|\psi\rangle$ by performing a unitary on his qubit accordingly.

> ***Exercise 2:*** Use CNOT, H, and X gates to come up with the quantum circuits to prepare the four EPR pairs in Eq. (4.2). Suppose you are now given an unknown EPR state:
> (a) How do you measure the state to infer the label?
> (b) As discussed above, depending on the measurement outputs from Alice, a unitary is performed on Bob's qubit to decode the message. List the decoding unitary in each case.

Exercise 3: Construct a function `teleport(Statevector)` that returns a `QuantumCircuit` that teleports the state from Alice to Bob, Test your protocol by teleporting a random state `random_statevector()` and **simulate** the circuit using `run()`. Compare your message states before and after the teleportation by `plot_bloch_multivector()`. (Hint: You may find the methods `QuantumCircuit.initialize()` and `c_if()` useful.)

4.2 Rabi Oscillation

Rabi oscillation is one of the basic principles behind single-qubit manipulation in quantum computing. It refers to the oscillation between $|0\rangle$ and $|1\rangle$ states in a two-level system under a periodic driving field. By adjusting the duration of the applied field (lasers or microwave pulses), one is able to perform desired single-qubit operation.
Suppose we have a two-level system (qubit), we apply a longitudinal field and a time-dependent transverse field $\mathbf{B} = B_z\hat{z} - B_x\cos(\omega t)\hat{x}$. The system has the Hamiltonian

$$\hat{H} = -\frac{\omega_0}{2}\hat{\sigma}_z + \omega_1\cos(\omega t)\hat{\sigma}_x \tag{4.3}$$

where $\omega_0 = \gamma B_z$ and $\omega_1 = \gamma B_x$ with γ being the gyromagnetic ratio. This system can be solved using the so-called *rotating wave approximation* (may sound familiar to some people?). The dynamics of a state $|\psi\rangle = \alpha(t)|0\rangle + \beta(t)|1\rangle$, initialized with $\alpha(0) = 1$, is given by

$$\begin{aligned} e^{-i\omega t/2}\alpha(t) &= \cos\left(\frac{\Omega t}{2}\right) - \frac{i\Delta}{\Omega}\sin\left(\frac{\Omega t}{2}\right) \\ e^{i\omega t/2}\beta(t) &= -\frac{i\omega_1}{\Omega}\sin\left(\frac{\Omega t}{2}\right) \end{aligned} \tag{4.4}$$

where $\Delta = \omega - \omega_0$ and $\Omega = \sqrt{\omega_1^2 + \Delta^2}$. Therefore, by choosing different t, some single-qubit operations could be performed. Next, we would like to use Qiskit to time-evolve our qubit and actually observe the dynamics in Eq. (4.4)!

Exercise 4: Compute the time-dependent population for state $|0\rangle$ and $|1\rangle$ from Eq. (4.4).

Exercise 5: Define a function `time_evolve(circ, t, dt, w, w0, w1)` that applies to the circuit `circ` the time-evolution operator $\hat{U}(t)$, which

is approximated by the time-dependent Trotterization

$$\hat{U}(t) = \mathcal{T}\exp\left(-i\int_0^t \hat{H}(t)dt\right) \approx \prod_{n=0}^{N-1}\exp\Big(-i\hat{H}(n\delta t)\delta t\Big)$$
$$\approx \prod_{n=0}^{N-1}\exp\left(\frac{i\omega_0\delta t}{2}\hat{\sigma}_z\right)\exp(-i\omega_1\delta t\cos(\omega n\delta t)\hat{\sigma}_x)$$

where $\delta t = t/N$ with N total time steps.
Measure the population of state $|1\rangle$ using 400 shots for:
(a) different t. You may use the following parameters, $w_1 = 2, w_0 = 25, w = 25.5$ for $t \in [0, 4]$. And fix the time step to be $\delta t = 0.05$.
(b) different detunings Δ (Vary $\omega \in [10, 40]$, at $t = \pi/\omega_1$)
Plot your measurement results against the solution from Exercise 4.
(c) Try also $\omega = \omega_0 = \omega_1 = 2$. Compare to the analytic solution and explain your findings.

The Trotterization you used above is in fact one of the fundamental concepts in digital quantum computing for doing tasks such as adiabatic quantum computation [5] and simulation of a quantum many-body system. On Day 2, you will get to explore the latter in more detail.
When you are actually running your simulation on a quantum computer, as we mentioned above, the results will be subject to various noise. Apart from readout errors, the interaction with the environment (decoherence) is another major source of errors. These types of errors become more prominent when your computation takes longer times. Let us simulate this effect in our Rabi oscillation experiment. Consider the so-called generalized amplitude damping quantum channel (GAD)

$$\Lambda(\rho) = \sum_{i=0}^{3} E_i \rho E_i^\dagger \tag{4.5}$$

where ρ is the density matrix of our system and the Kraus operators E_i are

$$E_0 = \sqrt{1-p}\begin{bmatrix}1 & 0\\ 0 & \sqrt{1-\gamma}\end{bmatrix}, \quad E_2 = \sqrt{p}\begin{bmatrix}\sqrt{1-\gamma} & 0\\ 0 & 1\end{bmatrix}$$
$$E_1 = \sqrt{1-p}\begin{bmatrix}0 & \sqrt{\gamma}\\ 0 & 0\end{bmatrix}, \qquad E_3 = \sqrt{p}\begin{bmatrix}0 & 0\\ \sqrt{\gamma} & 0\end{bmatrix}$$

This quantum channel gives an effective description of noise due to both spontaneous emission and finite temperature thermalization, where $p \in [0, 1]$ quantifies the probability for thermal excitation and $\gamma \in [0, 1]$ is the probability of losing energy to the environment (e.g. spontaneous emission). Let us try to simulate such noise!

Exercise 6: Add noise to the simulator `AerSimulator()` to perform the time evolution in Exercise 5 in a noisy environment. The GAD can be implemented by

`amplitude_damping_error(gamma, p)`.

Use the method `add_all_qubit_quantum_error()` to specify the noise so that Qiskit will automatically implement a GAD after every single-qubit gate in the circuit. You may use the values $p = 0.1$ and $\gamma = 0.02$ and the same parameters from 5(a), and plot your noisy result against what you obtained from Exercise 5(a). For a large enough time span, you should observe a decay time scale for $|1\rangle$ signal (sometimes called T_1).
Run the same measurements with a smaller time step $\delta t = 0.01$ and the same time span, what do you observe? Explain your finding.

Exercise 7: Repeat 5(a) using the fake machine backend

```
from qiskit.providers.fake_provider import FakeVigo
```

for time $t \in [0, 10]$. This performs the experiment using realistic noise models from the device Vigo. Compare your result to Exercise 6.

In this practical we will not attempt to mitigate such errors, in fact, error mitigation for such type is in general very subtle for more than a few qubits. It is however important to understand how these errors can affect your measurement result if your program is run on an actual device!

4.3 Single-qubit Tomography

By measuring the qubits in the computational basis, we only have access to the probability amplitude for each qubit state. To get the full information of the state, we need *quantum state tomography*. The basic idea is simple, any single-qubit density matrix can be written as a linear combination of Pauli matrices

$$\rho = \sum_{i=0}^{3} c_i \hat{\sigma}_i \tag{4.6}$$

where c_i are some real coefficients. $\hat{\sigma}_i$ are Pauli matrices with $\hat{\sigma}_0 = \mathbb{I}$, which form an orthogonal basis under the Hilbert Schmidt inner product $\langle A, B\rangle = Tr(A^\dagger B)$. To extract all the coefficients c_i from the measurement, we simply need to measure the system in all the orthogonal basis

$$c_i = \frac{1}{2}\langle \hat{\sigma}_i \rangle = \frac{1}{2} Tr(\hat{\sigma}_i \rho), \quad i = 1, 2, 3 \tag{4.7}$$

We do not need $i = 0$ because a physical density matrix must have unit trace, i.e. $c_0 = 1/2$.

Exercise 8: When you do a measurement on a quantum computer, you will only measure $|0\rangle$ or $|1\rangle$ (eigenstates of Z or $\mathbb{I}$), and hence their probabilities. However, this information is not enough to get $\langle\hat{\sigma}_x\rangle$ and $\langle\hat{\sigma}_y\rangle$. We need to instead measure $|0\rangle$ and $|1\rangle$ in different basis. Verify the following identities

$$X = HZH, \quad Y = SHZHS^\dagger \tag{4.8}$$

You can look up the definition of the phase gate S in Qiskit.
Use Eq. (4.8) to construct Qiskit circuits to measure $\langle\hat{\sigma}_x\rangle$, $\langle\hat{\sigma}_y\rangle$ and $\langle\hat{\sigma}_z\rangle$.

Exercise 9: Try the single-qubit tomograph method in Eq. (4.7), **measure** all the coefficients c_i from finite sampling, reconstruct the single-qubit density matrix for $(|0\rangle + e^{i\pi/4}|1\rangle)/\sqrt{2} = TH|0\rangle$ (T is a non-Clifford gate!). How do you extract the qubit state from the measured density matrix? Benchmark your result with the Qiskit built-in function `StateTomography()`.

Exercise 10: Use the same set of parameters as in Exercise 5(a), reconstruct the oscillating qubit state by tomography (either your custom one or Qiskit built-in) for $t \in [0, 2]$ and compare your result with Eq. (4.4) (both the complex coefficients $\alpha(t)$ and $\beta(t)$), do they agree? (Note that the states are equivalent up to an overall phase, you can set β to be real and compare α or vice versa).

The idea of state tomography indeed also generalizes to many-qubit states, which we will look into in more detail on day 2.

4.4 Variational Quantum Eigensolver (VQE)

A useful application of near-term quantum computers in studying quantum many-body systems is the so-called variational quantum eigensolver (VQE) [6, 7]. One of the standard techniques to approximate the ground state of a local Hamiltonian $\hat{H}$ is the variational method, namely you prepare a trial state $|\psi(\theta)\rangle$ parametrized by a set of parameters $\{\theta\}$, then we search for the state that minimizes the energy $\langle\psi(\theta)|\hat{H}|\psi(\theta)\rangle$.
However, in the classical computer, the evaluation of $\langle\psi(\theta)|\hat{H}|\psi(\theta)\rangle$ for a given set of parameters $\{\theta\}$ is exponentially hard for the general case of N qubits. The idea of VQE is to leave this difficulty to the quantum computer. Because one can directly measure the energy expectation $\langle\psi(\theta)|\hat{H}|\psi(\theta)\rangle$ by measuring expectation values of a list of Pauli strings which sum to $\hat{H}$! Hence VQE is a quantum-classical hybrid algorithm.
In a digital quantum computer, a simple VQE looks like this:

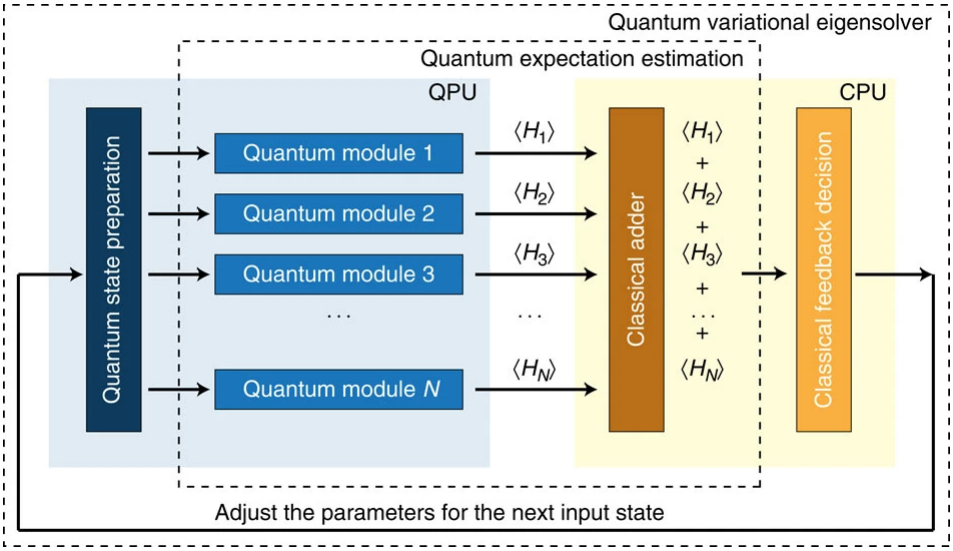

Figure 3: *Schematics of VQE. Here H_i are the terms that sum to the total Hamiltonian H, e.g. some Pauli strings. Figure taken from [6]*

1. Define a parametrized circuit ansatz that prepares the trial state $|\psi(\theta)\rangle$.
2. Run your favorite classical algorithm to find the minimum of $\langle\psi(\theta)|\hat{H}|\psi(\theta)\rangle$, but replace the evaluation step with measurement on the quantum computer.

Let us simulate this for a simple two-qubit example using Qiskit. Consider a two-qubit cluster state Hamiltonian

$$\hat{H} = -\hat{\sigma}_x^1\hat{\sigma}_z^2 - \hat{\sigma}_z^1\hat{\sigma}_x^2 \tag{4.9}$$

where $1, 2$ refers to the first and second qubits. Notice that the two terms appearing in the Hamiltonian commute with each other. Complete the following exercises.

Exercise 11: Let us consider the trial state of the form

$$|\psi(a,b)\rangle = \exp\big(ia(\hat{\sigma}_z^1 + \hat{\sigma}_z^2)\big)\exp\big(ib(\hat{\sigma}_x^1\hat{\sigma}_z^2 + \hat{\sigma}_z^1\hat{\sigma}_x^2)\big)\,|00\rangle \tag{4.10}$$

where a, b are some real parameters. Construct a parametrized circuit for this ansatz.

Exercise 12: In general we need to run some classical algorithm (e.g. gradient descent) to find the minimum energy state. This can in fact be done with Qiskit built-in modules.
But here for simplicity, just run a 20×20 grid search over the parameters space $a,\ b \in [0, \pi]$. , For each choice of parameters, **measure** the energy expectation $\langle\psi(a,b)|\hat{H}|\psi(a,b)\rangle$. Plot a heatmap (`imshow()`) for $\langle\psi(a,b)|\hat{H}|\psi(a,b)\rangle$ and find the best trial state. What is the approximated ground state energy?

(Hint: To measure the expectation values, you may either use the method you learned from Exercise 8 or `Estimator()`.)

Exercise 13: Verify that the state

$$|\psi\rangle = \frac{1}{\sqrt{2}}\left(|1-\rangle + |0+\rangle\right) \tag{4.11}$$

is the exact ground state of $\hat{H}$, where $|\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$. Compare your result from Exercise 12 to this exact solution.

Exercise 14: A nice thing about VQE is, it is to some extent insensitive to noise! To see this, repeat Exercise 12 but with noise added (you may use the noise model from Exercise 6 and try a larger noise with $p = 0.2$ and $\gamma = 0.1$). What is your VQE final state? How did the noise change your energy landscape?
(Hint: If you want to use `Estimator()`, you need to import the one from `aer.primitives` to add noise. Note that you have two-qubit gates in the circuit, you might need the method `tensor()` on the noise channel.)

References

[1] F. Arute, K. Arya, R. Babbush, *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[2] H.-L. Huang, D. Wu, D. Fan, and X. Zhu, "Superconducting quantum computing: a review," *Sci. China Inf. Sci.*, vol. 63, p. 180501, aug 2020.

[3] J. M. Gambetta, J. M. Chow, and M. Steffen, "Building logical qubits in a superconducting quantum computing system," *npj Quantum Inf.*, vol. 3, p. 2, dec 2017.

[4] B. Nachman, M. Urbanek, W. A. de Jong, and C. W. Bauer, "Unfolding quantum computer readout noise," *npj Quantum Inf.*, vol. 6, p. 84, dec 2020.

[5] R. Barends, A. Shabani, L. Lamata, *et al.*, "Digitized adiabatic quantum computing with a superconducting circuit," *Nature*, vol. 534, no. 7606, pp. 222–226, 2016.

[6] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien, "A variational eigenvalue solver on a photonic quantum processor," *Nat. Commun.*, vol. 5, p. 4213, sep 2014.

[7] C. Kokail, C. Maier, R. van Bijnen, T. Brydges, M. K. Joshi, P. Jurcevic, C. A. Muschik, P. Silvi, R. Blatt, C. F. Roos, and P. Zoller, "Self-verifying variational quantum simulation of lattice models," *Nature*, vol. 569, no. 7756, pp. 355–360, 2019.