Lesson 3: Sample-based quantum diagonalization

Sample-based quantum diagonalization (SQD) combines classical linear algebra and the power of quantum computing to diagonalize a Hamiltonian (matrix) and compute its eigenvalues and eigenvectors. Matrix diagonalization is an important mathematical operation as many problems in science, computation and optimization use the method.

The video below gives an overview of SQD, what determines its usefulness, and what makes it faster than many other approaches. The subsequent text gives more details.

1. Introduction and motivation

Consider the energy eigenvalue equation made famous by Schrödinger, as an example.

$$H|\psi\rangle = E|\psi\rangle$$

H is the Hamiltonian of a system, $|\psi\rangle$ is the wavefunction (also known as the eigenstate), and E is an eigenvalue. The eigenvalues of matrix H represent the energy levels of the system. For example, if the system is a molecule, the lowest eigenvalue represents the ground state energy of the molecule. In many problems, we are interested in ground state energy estimation.

By applying exact diagonalization techniques from linear algebra, we can diagonalize the full H matrix. However, the approach becomes computationally expensive (even impossible) as the matrix gets larger. For example, even for small chemical molecules, H can be prohibitively large (e.g., Hamiltonian for the N_2 molecule with a cc-PVDZ basis has a dimension of 65780×65780).

Fortunately, we do not always need all the eigenvalues and eigenvectors of a Hamiltonian H, and therefore, diagonalization of the full matrix is not required in many practical cases. For example, in case of ground state estimation, we are interested in the lowest eigenvalue and the corresponding eigenvector. This allows us to apply the concept of projection onto a (useful) subspace.

Consider an $N \times N$ matrix, H, where the complete vector space (Hilbert space) has a dimension of N (N is large). Next, we select a subspace ($\mathcal{S}$) - which is a subset of the full Hilbert space - of dimension M, where M is sufficiently small. After projecting H onto this subspace, the projected matrix (say, $H_{\mathcal{S}}$) will be smaller ($M \times M$). The smaller $H_{\mathcal{S}}$ can be diagonalized using a suitable classical numerical method, and generate eigenvalues and eigenvectors for that subspace.

Note that, the subspace must be in *support* of our target (e.g., ground) eigenstate. In other words, the projected Hamiltonian $H_{\mathcal{S}}$ must be in a subspace that includes the lowest eigenvalue.

2. Projection and Diagonalization

Consider we want to find the lowest eigenvalue and the corresponding eigenvector for the following 8×8 Hamiltonian matrix H.

$$H = \begin{bmatrix} 0.2235 & -0.0390 & -0.1035 & -0.0818 & 0.1746 & 0.1 \\ -0.0390 & 0.6621 & 0.0706 & -0.1964 & -0.0782 & 0.2 \\ -0.1035 & 0.0706 & 0.9961 & 0.1724 & 0.1067 & -0. \\ -0.0818 & -0.1964 & 0.1724 & -0.1773 & 0.1019 & -0. \\ 0.1746 & -0.0782 & 0.1067 & 0.1019 & 0.1418 & -0. \\ 0.1091 & 0.2619 & -0.2299 & -0.4778 & -0.1359 & 0.1 \\ 0.1165 & 0.1095 & -0.1817 & -0.1272 & -0.1793 & 0.1 \\ -0.0104 & 0.0029 & 0.1571 & -0.0414 & -0.0766 & 0.0 \end{bmatrix}$$

We will diagonalize the full matrix along with different projected versions ($H_{\mathcal{S}}$) for different subspaces to demonstrate the scalability and importance of the choice of subspace.

The ground state energy (minimum eigenvalue) of the matrix H is -0.5357 and the *exact* ground state wavefunction (eigenvector) is:

$$\text{GS}_{\text{exact}} = 0.8 * |011\rangle + 0.6 * |101\rangle.$$

i.e., the ground state of the matrix is spanned by two computational basis states (vectors) $|011\rangle$ and $|101\rangle$.

```python
import numpy as np
from scipy.linalg import eig

np.set_printoptions(precision=4, sign="-", suppress=T

H = np.array([
        [ 0.2235, -0.039 , -0.1035, -0.0818,  0.1746,
        [-0.0390,  0.6621,  0.0706, -0.1964, -0.0782,
        [-0.1035,  0.0706,  0.9961,  0.1724,  0.1067,
        [-0.0818, -0.1964,  0.1724, -0.1773,  0.1019,
        [ 0.1746, -0.0782,  0.1067,  0.1019,  0.1418,
        [ 0.1091,  0.2619, -0.2299, -0.4778, -0.1359,
        [ 0.1165,  0.1095, -0.1817, -0.1272, -0.1793,
        [-0.0104,  0.0029,  0.1571, -0.0414, -0.0766,
    ])
eigvals, eigvecs = eig(H)

print("Eigenvalues:")
print(eigvals)
print(f"Minimum eigenvalue: {eigvals.min()}")

print("\nEigenvectors (columns represent vectors):")
print(eigvecs)
print("\nEigenvector for the minimum eigenvalue (grou
print(eigvecs[:,np.argmin(eigvals)])
```

Output:

```
Eigenvalues:
[ 1.3039+0.j  0.947 +0.j  0.6405+0.j  0.3616+0.j -0.1321+0.
Minimum eigenvalue: (-0.5356560029438882+0j)

Eigenvectors (columns represent vectors):
[[-0.1288  0.0643  0.0806  0.8051  0.5612 -0.     -0.098  -
 [-0.2526 -0.5952 -0.585  -0.0092  0.1403 -0.      0.1985 -
 [ 0.7086 -0.5794  0.0139  0.1509 -0.0416  0.     -0.3041
 [ 0.2966  0.1557  0.0838 -0.1081  0.1936  0.8     0.0127 -
 [ 0.1941  0.1449 -0.0954  0.5395 -0.6716  0.      0.3535 -
 [-0.3954 -0.2076 -0.1118  0.1441 -0.258   0.6    -0.017
 [-0.3657 -0.2625  0.468   0.0626 -0.3088  0.     -0.5504 -
 [-0.0418 -0.3856  0.6352 -0.0394  0.1146 -0.      0.6559

Eigenvector for the minimum eigenvalue (ground state)
[-0.  -0.   0.   0.8  0.   0.6  0.  -0. ]
```

Next, we will project the matrix H onto different subspaces and check if we can get the exact ground state. In particular, we will project the matrix onto a subspace spanned by:

1. exact ground state vectors ($|011\rangle$ and $|101\rangle$).
2. vectors that excludes some or all exact ground state vectors (e.g., $|000\rangle$, $|011\rangle$, and $|110\rangle$).
3. vectors that include both exact ground state and out-of-ground state (but not all possible vectors in the Hilbert space).

2.1 Case-1: Subspace includes ground state

Suppose we want to project H in a subspace ($\mathcal{S}$) spanned by two vectors $x_1 = |011\rangle$ and $x_2 = |101\rangle$. The projected Hamiltonian is defined by:

$$H_{\mathcal{S}} = \begin{bmatrix} \langle x1|H|x1\rangle & \langle x1|H|x2\rangle \\ \langle x2|H|x1\rangle & \langle x2|H|x2\rangle \end{bmatrix}$$

```
x1 = np.zeros(8)
x1[3] = 1 # binary 011 is 3 in decimal. |011> = |3> -

x2 = np.zeros(8)
x2[5] = 1 # binary 101 is 5 in decimal

Hs = np.array(
    [
```

```
            [x1 @ H @ x1.T, x1 @ H @ x2.T],
            [x2 @ H @ x1.T, x2 @ H @ x2.T]
        ]
    )
    print(Hs)
```

Output:

```
[[-0.1773 -0.4778]
 [-0.4778  0.1014]]
```

```
eigvals, eigvecs = eig(Hs)
print(f"Minimum eigenvalue: {eigvals.min()}")
print(f"Eigenvector for minimum eigenvalue: {eigvecs[:
```

Output:

```
Minimum eigenvalue: (-0.5356560000642949+0j)
Eigenvector for minimum eigenvalue: [-0.8 -0.6]
```

We can make several key observations here.

- As we spanned the subspace with two vectors, the dimension of the projected matrix ($H_{\mathcal{S}}$) is 2×2, which is smaller than the full matrix H (8×8).
- The minimum eigenvalue of the projected matrix matches the exact ground state's eigenvalue.
- The values in the `eigvecs` variable denotes the amplitude of subspace spanning vectors, and using them we can reconstruct the eigenstate (ground state). In this case, we end with the exact ground state (up to a global phase):

$$|\psi\rangle = -(0.8|011\rangle + 0.6|101\rangle)$$

2.2 Case-2: Subspace excludes some or all ground state vectors

Next, we project H onto a subspace spanned by three vectors $x_1 = |000\rangle$, $x_2 = |011\rangle$, and $x_3 = |110\rangle$. We deliberately choose the vectors

such that it excludes a ground state vector ($|101\rangle$). The projected Hamiltonian is defined by:

$$H_S = \begin{bmatrix} \langle x1|H|x1\rangle & \langle x1|H|x2\rangle & \langle x1|H|x3\rangle \\ \langle x2|H|x1\rangle & \langle x2|H|x2\rangle & \langle x2|H|x3\rangle \\ \langle x3|H|x1\rangle & \langle x3|H|x2\rangle & \langle x3|H|x3\rangle \end{bmatrix}$$

```
x1 = np.zeros(8)
x1[0] = 1

x2 = np.zeros(8)
x2[3] = 1

x3 = np.zeros(8)
x3[6] = 1

Hs = np.array(
    [
        [x1 @ H @ x1.T, x1 @ H @ x2.T, x1 @ H @ x3.T]
        [x2 @ H @ x1.T, x2 @ H @ x2.T, x2 @ H @ x3.T]
        [x3 @ H @ x1.T, x3 @ H @ x2.T, x3 @ H @ x3.T]
    ]
)
print(Hs)
```

Output:

```
[[ 0.2235 -0.0818  0.1165]
 [-0.0818 -0.1773 -0.1272]
 [ 0.1165 -0.1272  0.4227]]
```

```
eigvals, eigvecs = eig(Hs)
print(f"Minimum eigenvalue: {eigvals.min()}")
```

Output:

```
Minimum eigenvalue: (-0.21108858736702363+0j)
```

The eigenvalue -0.2111 in this case does not match the minimum eigenvalue -0.5357 of the full Hamiltonian. The key observation here is:

if we project onto a subspace that excludes basis states in our target (ground) state - either partially or completely - the estimated ground state will be different than the exact one.

2.3 Case-3: Subspace includes both ground state and non-ground state vectors

Next, we show a case where the subspace is spanned by vectors that include exact ground state vectors along with unwanted vectors. Suppose our subspace is spanned by $x_1 = |011\rangle$, $x_2 = |101\rangle$ (present in the exact ground state), and $x_3 = |111\rangle$ (absent in the exact ground state).

```
x1 = np.zeros(8)
x1[3] = 1

x2 = np.zeros(8)
x2[5] = 1

x3 = np.zeros(8)
x3[7] = 1

Hs = np.array(
    [
        [x1 @ H @ x1.T, x1 @ H @ x2.T, x1 @ H @ x3.T]
        [x2 @ H @ x1.T, x2 @ H @ x2.T, x2 @ H @ x3.T]
        [x3 @ H @ x1.T, x3 @ H @ x2.T, x3 @ H @ x3.T]
    ]
)
print(Hs)
```

Output:

```
[[-0.1773 -0.4778 -0.0414]
 [-0.4778  0.1014  0.0552]
 [-0.0414  0.0552  0.4456]]
```

```
eigvals, eigvecs = eig(Hs)
print(f"Minimum eigenvalue: {eigvals.min()}")
print(f"Eigenvector for minimum eigenvalue: {eigvecs[:
```

Output:

```
Minimum eigenvalue: (-0.5356560000646104+0j)
Eigenvector for minimum eigenvalue: [ 0.8  0.6 -0. ]
```

In this case, we again get -0.5357 as the minimum eigenvalue which matches with the full matrix (i.e., the exact ground state). Another interesting result is the amplitude of x_3 returned by the projection and diagonalization process. The amplitude is 0, and when we reconstruct the wavefunction (eigenstate) with computed amplitudes and vectors, we get:

$$|\psi\rangle = 0.8|011\rangle + 0.6|101\rangle + 0.0|111\rangle = 0.8|011\rangle + 0.6|101\rangle \text{ (exac}$$

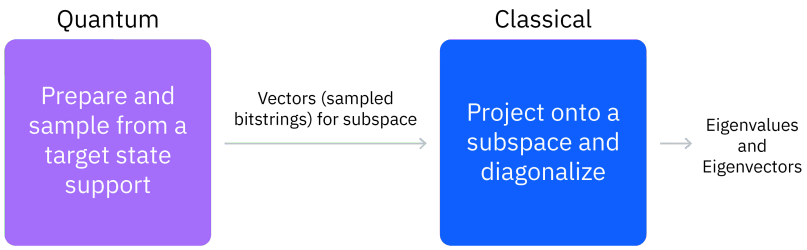
Thus, even if our subspace includes some non-target vectors (along with the full set of target vectors), we can compute the correct eigenvalue and eigenstate as the projection and diagonalization process filters out non-target vectors by setting their amplitudes to 0. This property of SQD provides an inherent noise tolerance.

3. Role of quantum in SQD

Above analyses establish the importance of subspace spanning vectors, which must be in *support of the target state*. This raises an important question: **How do we choose vectors with target state support for subspace construction?**

This is where **quantum computers** come into play. The quantum-classical synergy works as follows in the SQD paradigm:

1. Using a suitable quantum circuit, we try to prepare a state on a quantum computer that will generate basis states on which the target wavefunction (e.g., ground state) has significant support. The sampled basis states (bitstrings) will span the subspace for Hamiltonian projection.
2. A classical computer projects the Hamiltonian on to the subspace (spanned by samples/vectors from the quantum computer) and diagonalizes it to compute eigenvalues and eigenvectors using suitable numerical methods.

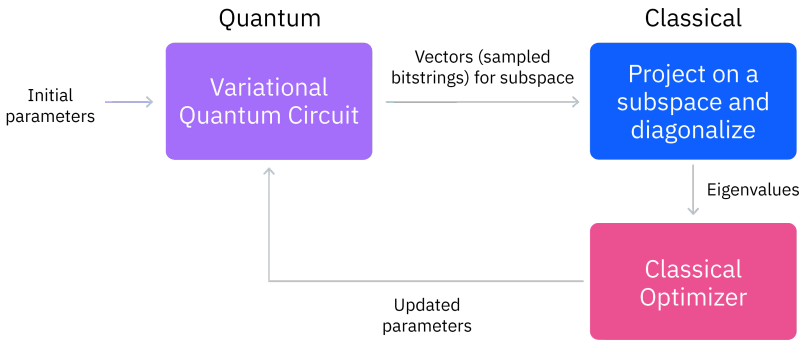


There can be several ways we can prepare such a quantum state, and they can be variational or non-variational depending on the problem.

In next two lessons, we will show two specific examples of preparing states and sampling from them.

1. In Lesson 4, we will use a parameterized local unitary coupled Jastrow (LUCJ) ansatz to generate samples for a chemistry problem (ground state energy estimation of N_2 molecule). We will initialize the LUCJ ansatz with parameters from classical coupled cluster singles and doubles (CCSD) computation.
2. In Lesson 5, we will sample from Krylov basis states to span the subspace for a condensed matter physics problem. This approach is non-variational in nature.

Besides the problem-specific approaches above, a generic approach for state preparation involves a variational ansatz, where we will iteratively update ansatz parameters using a classical optimizer.



3.1 Notes on ground state support

Let us explain the concept of *ground state support* more. Ground state support can be defined as the set of basis states where ground state has non-zero amplitude (up to a cutoff threshold).

Suppose the exact ground state of a 3-qubit problem is

$$|\psi\rangle = \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle$$

If we sample the above state, we should get a set of computational basis states $\{|000\rangle, |111\rangle\}$ (other computational basis states have zero amplitude in the ground state, and therefore, ideally will not appear during sampling).

In practice, we do not need to prepare the exact ground state as sampling many other states can give us the same set of vectors. For example:

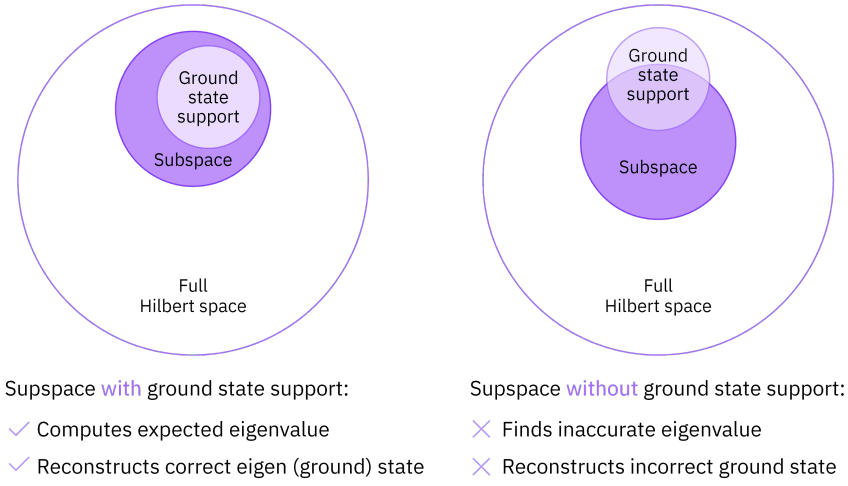
$$|\psi_a\rangle = 0.8|000\rangle + 0.6|111\rangle \xrightarrow{\text{Sampling}} \{|000\rangle, |111\rangle\} \quad (1)$$

$$|\psi_b\rangle = \frac{1}{2}|000\rangle + \frac{\sqrt{3}}{2}|111\rangle \xrightarrow{\text{Sampling}} \{|000\rangle, |111\rangle\} \quad (2)$$

$$|\psi_c\rangle = \frac{1}{2}|000\rangle + \frac{1}{2}|111\rangle + \frac{1}{\sqrt{2}}|101\rangle \xrightarrow{\text{Sampling}} \{|000\rangle, |101\rangle, |111\rangle\} \quad (3)$$

Preparing and sampling from any of the above states will generate vectors that have non-zero amplitude in the ground state, and all of them

qualify as to have *ground state support*. Note that, sampling $|\psi_c\rangle$ includes one extra vector $|101\rangle$ which has 0 amplitude in the exact ground state. However, earlier we have shown that including such vectors in the subspace is not problematic as the projection and diagonalization operation sets the amplitude of unwanted vectors to 0, and we can get expected eigenvalue and reconstruct the correct eigenstate.



Thus, preparing and sampling from the exact ground state is not necessary. In fact, doing so can be difficult as the exact ground state is not known *a priori*, and it is often beneficial to **not** prepare and sample from the exact ground state, especially if the wavefunction (state) is skewed with some basis states having very high probabilities. Consider the following wavefunction:

$$|\psi\rangle = 0.7|000\rangle - 0.7|010\rangle + 0.1|101\rangle - 0.01|111\rangle$$

This is a skewed wavefunction where basis states $|000\rangle$ and $|010\rangle$ have much larger amplitudes compared to $|101\rangle$ and $|111\rangle$. When sampled, we will get $|000\rangle$ and $|010\rangle$ more frequently ($\text{sampling probability} = |\text{amplitude}|^2 \approx 49\%$ for $|000\rangle$ and $|010\rangle$ each, $\approx 1\%$ for $|101\rangle$, and $\approx 0.01\%$ for $|111\rangle$). With finite sampling budget (*shots*), it is highly likely that our sampled set only contains $|000\rangle$ and $|010\rangle$. As shown earlier, if we span the subspace with such a set with missing vectors, we will not be able to find the true minimum eigenvalue. Therefore, it will be beneficial (and necessary) to sample from a state with ground state support.

3.2 A case against uniform sampling

It can be tempting to draw samples from a uniform distribution to span the subspace. While it may work for small problems, it will start to fail for larger and more practical problems. For large problems with many qubits, the Hilbert space can be prohibitively large. For example, a 32-qubit Hilbert space has more than 4 billion possible basis vectors ($2^{32} = 4,294,967,296$). If we uniformly sample from that space with finite sample budget (say 10000 vectors to keep the diagonalization process feasible), the subspace may exclude vectors with ground state support more often as the process will be random. **Therefore, we need a systematic way to sample from ground state support leveraging quantum circuits.**

4. SQD and Sparsity of the wavefunction

The gap between full Hilbert space and feasible subspace dimensions, brings another important aspect of SQD, and that is the sparsity of the wavefunction. The SQD approach works well for sparse or concentrated wavefunctions where a small fraction of basis states has non-negligible amplitudes. There are two reasons behind it:

1. If the wavefunction is broad (i.e., many basis states have non-negligible amplitudes), and we miss out including vectors with target state support in the subspace, we may end up with incorrect eigenvalues and eigenvectors.
2. To avoid the above issue, we need to include many vectors in the subspace. However, the dimension of the projected Hamiltonian is directly related to the subspace dimension. A larger subspace will mean larger Hamiltonian, which may become infeasible to diagonalize.

We showcase the issue with the following matrix (H_{new}). The lowest eigenvalue of the H_{new} is -2.2081, and the corresponding wavefunction (eigenstate) is broad:

$$|\psi\rangle = \frac{|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |1}{\sqrt{8}}$$

```
H_new = np.array([
    [-0.958 ,  0.1853, -0.2663, -0.3875, -0.0524, -0.
    [ 0.1853, -0.4081, -0.8549, -0.2312,  0.0615, -0.
    [-0.2663, -0.8549, -0.6929, -0.0063, -0.0478, -0.
    [-0.3875, -0.2312, -0.0063, -0.4468, -0.6301, -0.
```

```
    [-0.0524,  0.0615, -0.0478, -0.6301, -0.6664, -0.
    [-0.3779, -0.2493, -0.0236, -0.4627, -0.1514, -0.
    [-0.0145, -0.3804, -0.2494, -0.1188, -0.3571,  0.
    [-0.3369, -0.3312, -0.0669,  0.0753, -0.3644,  0.
])
```

No output produced

```
eigvals, eigvecs = eig(H_new)
print(f"Minimum eigenvalue: {eigvals.min()}")
print(f"Eigenvector for minimum eigenvalue: {eigvecs[:
```

Output:

```
Minimum eigenvalue: (-2.208137504726665+0j)
Eigenvector for minimum eigenvalue: [0.3536 0.3536 0.3536 0
```

Suppose we project the H_{new} onto a subspace spanned by four vectors: $|000\rangle$, $|010\rangle$, $|101\rangle$, and $|110\rangle$ and compute the eigenvalue.

```
x1 = np.zeros(8)
x1[0] = 1

x2 = np.zeros(8)
x2[2] = 1

x3 = np.zeros(8)
x3[5] = 1

x4 = np.zeros(8)
x4[6] = 1

H_new_s = np.array(
    [
        [x1 @ H_new @ x1.T, x1 @ H_new @ x2.T, x1 @ H
        [x2 @ H_new @ x1.T, x2 @ H_new @ x2.T, x2 @ H
        [x3 @ H_new @ x1.T, x3 @ H_new @ x2.T, x3 @ H
        [x4 @ H_new @ x1.T, x4 @ H_new @ x2.T, x4 @ H
    ]
)
print(H_new_s)
```

Output:

```
[[-0.958  -0.2663 -0.3779 -0.0145]
 [-0.2663 -0.6929 -0.0236 -0.2494]
 [-0.3779 -0.0236 -0.9605  0.0137]
 [-0.0145 -0.2494  0.0137 -1.1449]]
```

```
eigvals, eigvecs = eig(H_new_s)
print(f"Minimum eigenvalue: {eigvals.min()}")
```

Output:

```
Minimum eigenvalue: (-1.426655234058667+0j)
```

The above example shows that when the wavefunction is broad, and we do not include basis states in the subspace, the eigenvalue computation becomes incorrect.

5. SQD vs. VQE

As noted earlier, SQD may need a variational quantum circuit and iterative parameter updates to prepare and sample from ground state support. As this iterative parameter update routine is similar to VQE, one can ask how these methods are different, and what are the advantages of SQD over VQE? In this section, we compare the methods and discuss advantages of SQD with a N_2 molecule described with minimal basis set (`sto-3g`) as an example.

	VQE	SQD
Measurement overhead	**Many Pauli terms, many measurement circuits:** The Hamiltonian for the molecule has 2951 unique Pauli terms. As the Pauli terms can contain X and Y terms, and typical quantum measurements are done in the Z-basis,	In SQD, we do not need different measurement circuits for each grouped Pauli terms. Typically, we measure a single circuit for a fixed number of shots. While we may set the number of shots to a large value depending on the problem, the

	VQE	SQD
	we need measurement basis change to evaluate those terms. When optimized for measurements, the 2951 terms can be grouped into 1187 groups, where each group can be evaluated using a single circuit. Thus, we need at least 1187 unique circuits to evaluate all Pauli terms. **Many shots per circuit for tighter variance**. Again, the evaluated expectation value of each Pauli term has a variance attached to them which depends inversely on the $\sqrt{shots}$. Therefore, to accurately estimate each term, we need to allocate many shots per circuit. For example, to achieve chemical accuracy (1 kcal/mol), typically we need shots in the order of 10^5-10^7 per circuit. Thus, VQE needs many measurement circuits and each circuit with certain number of shots. For practical cases, this measurement overhead can be restrictive.	overhead remains much smaller than VQE. Also, the energy estimations using diagonalization process are exact, meaning the computed eigenvalues are exact in that subspace and does not have a variance attached to them like VQE. (In case of Krylov basis state sampling (Lesson 5), we need to measure multiple circuits, but the number circuit remains much smaller than VQE).
Estimated energy bound	In VQE, energy estimations are not bounded and can be lower than true minimum values due to noise.	The energy estimation process in SQD always produces upper bound to ground state energy and estimated energy will never be lower than the true ground state energy.
Noise tolerance	VQE energy estimation is susceptible to noise from pre-fault-tolerant quantum computers.	SQD has inherent tolerance to noise. Pre-fault-tolerant quantum computers can produce noisy samples. Even if we include those samples in the subspace, the subsequent diagonalization can suppress those samples

VQE	SQD
	by setting their amplitudes to zero. Also, we will discuss a method called *configuration recovery* in relation to SQD that improves SQD noise tolerance further.

6. Summary

1. In SQD, a quantum computer generate samples and a classical computer projects a Hamiltonian onto a subspace spanned by the samples and diagonalizes it to compute eigenvalues and eigenvectors.
2. The generated samples should be from the target (ground) state support.
3. Depending on the problem, the quantum state preparation and sample generation flow can be iterative or non-iterative.
4. SQD works best for sparse wavefunctions. A broad wavefunction will