SKKU 2025 Quantum Challenge : Measuring Berry phase in quantum computing

Sangyoon Woo

November 14, 2025

Part 1: What is Berry phase?

Question 1.1

According to the problem, the quantum state starting from the initial eigenstate $|E(\lambda(0))\rangle$ is expressed as Eq. 1 in time t.

$$|\psi(t)\rangle = u(t)\,|E(\lambda(t))\rangle\,. \tag{1}$$

And the Schrödinger equation for the time-varying Hamiltonian is expressed as Eq. 2.

$$i\frac{d}{dt}\,|\psi(t)\rangle = H(\lambda(t))\,|\psi(t)\rangle \tag{2}$$

Thus, we can substitute Eq. 1 for the quantum state $|\psi(t)\rangle$ in Eq. 2 as follows:

$$i\frac{d}{dt}\Big(u(t)\,|E(\lambda(t))\rangle\Big) = H(\lambda(t))\,|\psi(t)\rangle\,. \tag{3}$$

$$i\dot{u}(t)\,|E(\lambda(t))\rangle + iu(t)\frac{d}{dt}\,|E(\lambda(t))\rangle = H(\lambda(t))u(t)\,|E(\lambda(t))\rangle\,. \tag{4}$$

Also, in the right-hand side, $|E(\lambda(t))\rangle$ is an eigenvector of the Hamiltonian $H(\lambda(t))$, whose eigenvalue at this time is $E(\lambda(t))$. Therefore, the expression can be rewritten as:

$$i\dot{u}(t)\,|E(\lambda(t))\rangle + iu(t)\frac{d}{dt}\,|E(\lambda(t))\rangle = u(t)E(\lambda(t))\,|E(\lambda(t))\rangle\,. \tag{5}$$

Now, to project along the direction of the vector $|E(\lambda(t))\rangle$, let us multiply both sides by $\langle E(\lambda(t))|$ from the left. Then we obtain:

$$\begin{aligned}\langle E(\lambda(t))|\; i\dot{u}(t)\,|E(\lambda(t))\rangle + \langle E(\lambda(t))|\; iu(t)\frac{d}{dt}\,|E(\lambda(t))\rangle \\ = \langle E(\lambda(t))|\; u(t)E(\lambda(t))\,|E(\lambda(t))\rangle\,.\end{aligned} \tag{6}$$

Since $\langle\psi|\psi\rangle = 1$, the equation can be simplified as follows:

$$i\dot{u}(t) + iu(t)\,\langle E(\lambda)|\,\frac{d}{dt}\,|E(\lambda)\rangle = u(t)E(\lambda). \tag{7}$$

Now, by rearranging the terms with respect to $u(t)$, we obtain:

$$\frac{\dot{u}(t)}{u(t)} = -iE(\lambda(t)) - \langle E(\lambda(t))| \frac{d}{dt} |E(\lambda(t))\rangle . \tag{8}$$

Since we have already rearranged the equation for $u(t)$, we can integrate Eq. 8 with respect to time to obtain $u(t)$. By integrating, we have:

$$\ln u(t) = -i \int_0^t E(\lambda(t'))\, dt' - \int_0^t \langle E(\lambda(t'))| \frac{d}{dt'} |E(\lambda(t'))\rangle \, dt'. \tag{9}$$

Therefore, $u(t)$ can be calculated as follows:

$$u(t) = \exp\left[-i \int_0^t E(\lambda(t'))\, dt' - \int_0^t \langle E(\lambda(t'))| \frac{d}{dt'} |E(\lambda(t'))\rangle \, dt'\right]. \tag{10}$$

Now, let us take a closer look at the exponent of e. The right-hand side of the exponent can be written as:

$$- \int_0^t \langle E(\lambda(t'))| \frac{d}{dt'} |E(\lambda(t'))\rangle \, dt'. \tag{11}$$

Here, the state $|E(\lambda(t))\rangle$ does not depend directly on t, but rather on the parameter $\lambda(t)$. Since $\frac{d}{dt'} = \frac{d\lambda}{dt'}\frac{d}{d\lambda}$, we can rewrite the above expression as a path integral instead of a time integral:

$$- \int_{\lambda(0)}^{\lambda(t)} \langle E(\lambda)| \frac{d}{d\lambda} |E(\lambda)\rangle \, d\lambda. \tag{12}$$

Since the Berry connection $A(\lambda)$ is defined as in Eq. 13, we can rewrite Eq. 12 using $A(\lambda)$ as Eq. 14 in the following form:

$$A(\lambda) = -i \langle E(\lambda)| \frac{d}{d\lambda} |E(\lambda)\rangle . \tag{13}$$

$$-i \int_{\lambda(0)}^{\lambda(t)} A(\lambda)\, d\lambda. \tag{14}$$

The above rewritten expression corresponds exactly to the Berry phase θ_{Berry}. This demonstrates that the Berry phase arises because the eigenstate $|E(\lambda(t))\rangle$ itself rotates as time progresses.

Question 1.2

The Hamiltonian $H(\vec{h}(t))$ is given by $H(\vec{h}(t)) = \vec{h}(t) \cdot \vec{\sigma} = h_x X + h_y Y + h_z Z$, which can be expressed in matrix form as follows:

$$H(\vec{h}(t)) = \begin{pmatrix} h_z & h_x - ih_y \\ h_x + ih_y & -h_z \end{pmatrix}. \tag{15}$$

We need to use this matrix to calculate the eigenvalues and eigenstates. Before that, we must solve $\det(H - EI) = 0$ to find the allowed eigenvalues E.

First, expressing $H - EI$ in matrix form gives:

$$H - EI = \begin{pmatrix} h_z - E & h_x - ih_y \\ h_x + ih_y & -h_z - E \end{pmatrix}. \tag{16}$$

Then, the determinant $\det(H - EI)$ becomes

$$\det(H - EI) = (-h_z^2 + E^2) - (h_x^2 + h_y^2) = 0. \tag{17}$$

Hence, we obtain

$$E^2 = h_x^2 + h_y^2 + h_z^2 = |\vec{h}|^2. \tag{18}$$

Therefore, the eigenvalues are $E = \pm|\vec{h}|$, which indicates that there exist two distinct energy levels.

Now, using the eigenvalues $E_0 = -|\vec{h}|$ and $E_1 = |\vec{h}|$, we can obtain the corresponding eigenvectors $|E_0\rangle$ and $|E_1\rangle$. The eigenvectors can be found by solving the eigenvalue equation.

Let us first find $|E_0\rangle$. Assuming $|E_0\rangle \propto \begin{pmatrix} a \\ b \end{pmatrix}$, the eigenvalue equation $(H - E_0 I)\,|E_0\rangle = 0$ can be written in matrix form as:

$$\begin{pmatrix} h_z + |\vec{h}| & h_x - ih_y \\ h_x + ih_y & -h_z + |\vec{h}| \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = 0. \tag{19}$$

Expanding this matrix equation gives the following two linear equations:

$$\begin{aligned} (h_z + |\vec{h}|)a + (h_x - ih_y)b = 0, \\ (h_x + ih_y)a + (-h_z + |\vec{h}|)b = 0. \end{aligned} \tag{20}$$

From the first equation in Eq. (20), we can express b in terms of a as:

$$b = -\frac{h_z + |\vec{h}|}{h_x - ih_y}\, a. \tag{21}$$

Therefore, the eigenvector $|E_0\rangle$ can be written as:

$$|E_0\rangle \propto a \begin{pmatrix} 1 \\ -\dfrac{h_z + |\vec{h}|}{h_x - ih_y} \end{pmatrix} \propto N \begin{pmatrix} h_x - ih_y \\ -h_z - |\vec{h}| \end{pmatrix}, \tag{22}$$

where N is the normalization constant.

Since the inner product of a quantum state satisfies $\langle E_0|E_0\rangle = 1$, we must normalize the eigenstate.

$$\begin{aligned} \langle E_0|E_0\rangle &= N^2 \left(|h_x - ih_y|^2 + |-h_z - |\vec{h}||^2 \right) \\ &= N^2 \left((h_x^2 + h_y^2) + (h_z + |\vec{h}|)^2 \right) \\ &= N^2 \left(h_x^2 + h_y^2 + h_z^2 + 2h_z|\vec{h}| + |\vec{h}|^2 \right) \\ &= 2N^2|\vec{h}|(h_z + |\vec{h}|) = 1. \end{aligned} \tag{23}$$

From the above relation, the normalization constant N is given by:

$$N = \frac{1}{\sqrt{2|\vec{h}|(h_z + |\vec{h}|)}}. \tag{24}$$

Therefore, the normalized eigenstate $|E_0(\vec{h})\rangle$ is expressed as:

$$|E_0(\vec{h})\rangle = \frac{1}{\sqrt{2|\vec{h}|(h_z + |\vec{h}|)}} \begin{pmatrix} h_x - ih_y \\ -h_z - |\vec{h}| \end{pmatrix}. \tag{25}$$

The other eigenstate $|E_1(\vec{h})\rangle$ can be obtained in the same way by replacing $E_0 = -|\vec{h}|$ with $E_1 = +|\vec{h}|$, giving:

$$|E_1(\vec{h})\rangle = \frac{1}{\sqrt{2|\vec{h}|(-h_z + |\vec{h}|)}} \begin{pmatrix} h_x - ih_y \\ -h_z + |\vec{h}| \end{pmatrix}. \tag{26}$$

Thus, we have successfully derived the eigenvalues E_0, E_1 and their corresponding eigenvectors for the Hamiltonian H.

Question 1.3

First, let us summarize the initial state. From the problem statement, the initial Hamiltonian at the starting Bloch vector $\vec{h}(\lambda = 0) = (1, 0, 0)$ is given by

$$H(0) = 1 \cdot X + 0 \cdot Y + 0 \cdot Z = X. \tag{27}$$

which means that

$$|\psi(0)\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}. \tag{28}$$

The overall computational structure is identical for all trajectories; only the Bloch vector $\vec{h}(t)$ differs among them. Therefore, before proceeding further, we first determine the time-dependent Bloch vector $\vec{h}(t)$ for the four trajectories shown in Fig. 1.

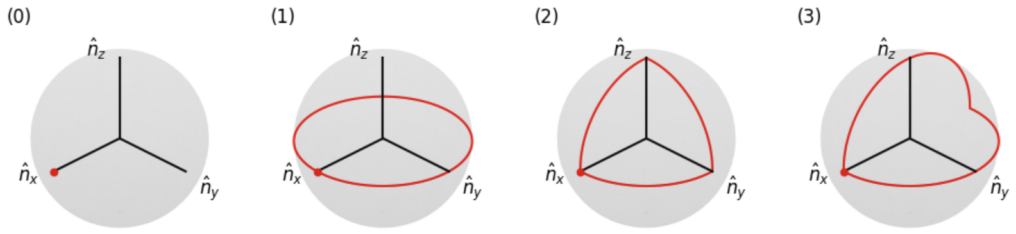


Figure 1: The four trajectories presented in the problem.

Before analyzing the trajectories, the Bloch sphere can be expressed in spherical coordinates as follows:

$$\vec{h}(\theta,\phi) = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{pmatrix} \tag{29}$$

Here, θ denotes the polar angle measured from the z-axis, and ϕ represents the azimuthal angle in the xy-plane. The trajectory can thus be parameterized using these two angles.

The first trajectory corresponds to a fixed point along the x-axis. This implies that for all times t, the Bloch vector remains constant as $\vec{h}(t) = (1, 0, 0)$, which is equivalent to expressing it in terms of λ as:

$$\vec{h}_0(\lambda) = (1, 0, 0) \tag{30}$$

Since the vector does not change over time, this case is identical for all λ.

The second trajectory corresponds to a circular motion on the xy-plane. Here, $\theta = \pi/2$ is fixed, and only ϕ varies from 0 to 2π over time. Substituting these values into Eq. (29) gives:

$$\vec{h}_1(\phi) = \begin{pmatrix} \sin(\pi/2)\cos\phi \\ \sin(\pi/2)\sin\phi \\ \cos(\pi/2) \end{pmatrix} = \begin{pmatrix} \cos\phi \\ \sin\phi \\ 0 \end{pmatrix} \tag{31}$$

Rewriting this in terms of λ:

$$\vec{h}_1(\lambda) = \begin{pmatrix} \cos(2\pi\lambda) \\ \sin(2\pi\lambda) \\ 0 \end{pmatrix} \tag{32}$$

The third trajectory forms a closed loop passing sequentially through $(1, 0, 0) \to (0, 1, 0) \to (0, 0, 1) \to (1, 0, 0)$. Thus, the trajectory is divided into three segments over $\lambda \in [0, 1]$. The Bloch vector $\vec{h}_2(\lambda)$ is expressed as:

$$\vec{h}_2(\lambda) = \begin{cases} (\cos\frac{\pi t}{2}, \sin\frac{\pi t}{2}, 0), & t = 3\lambda, \quad 0 \le \lambda < \frac{1}{3} \\ (0, \cos\frac{\pi t}{2}, \sin\frac{\pi t}{2}), & t = 3\lambda - 1, \quad \frac{1}{3} \le \lambda < \frac{2}{3} \\ (\sin\frac{\pi t}{2}, 0, \cos\frac{\pi t}{2}), & t = 3\lambda - 2, \quad \frac{2}{3} \le \lambda \le 1 \end{cases} \tag{33}$$

This trajectory starts from $(1, 0, 0)$, passes sequentially through $(0, 1, 0)$ and $(0, 0, 1)$, and returns to its original position $(1, 0, 0)$.

Finally, the last trajectory is also divided into two segments, and the path can be defined accordingly. By considering the range of λ, the Bloch vector is expressed as:

$$\vec{h}_3(\lambda) = \begin{cases} (\cos(\pi t),\, \sin(\pi t),\, 0), & t = 2\lambda, \quad 0 \le \lambda < \frac{1}{2} \\ (-\cos(\pi t),\, 0,\, \sin(\pi t)), & t = 2\lambda - 1, \quad \frac{1}{2} \le \lambda \le 1 \end{cases} \tag{34}$$

Each of the Bloch vectors derived above can be implemented in code form as shown in Listing 1.

Listing 1: Functions that generate four trajectories.

```
# (0)
def h_vec_0(lmbd):
    return np.array([1, 0, 0])

# (1)
def h_vec_1(lmbd):
    return np.array([
        np.cos(2 * np.pi * lmbd),
        np.sin(2 * np.pi * lmbd),
        0
    ])

# (2)
def h_vec_2(lmbd):
    if lmbd < 1/3:
        t = 3 * lmbd
        return np.array([
            np.cos(np.pi/2 * t),
            np.sin(np.pi/2 * t),
            0
        ])
    elif lmbd < 2/3:
        t = 3 * lmbd - 1
        return np.array([
            0,
            np.cos(np.pi/2 * t),
            np.sin(np.pi/2 * t)
        ])
    else:
        t = 3 * lmbd - 2
        return np.array([
            np.sin(np.pi/2 * t),
            0,
            np.cos(np.pi/2 * t)
        ])

# (3)
def h_vec_3(lmbd):
    if lmbd < 0.5:
        t = 2 * lmbd
        return np.array([
            np.cos(np.pi * t),
            np.sin(np.pi * t),
            0
        ])
    else:
        t = 2 * lmbd - 1
        return np.array([
            -np.cos(np.pi * t),
            0,
            np.sin(np.pi * t)
        ])
```

Using the time-dependent Bloch vectors, the Schrödinger equation is numerically integrated at each timestep by applying a unitary operator. The integrated result is compared with the initial state through the inner product, allowing the cumulative change in the Berry phase to be computed. The code that performs this calculation and visualizes the results is shown in Listing 2, and the simulation results for each Bloch vector are presented in Fig. 2 and Outputs 1–4.

Listing 2: Functions that generate four trajectories.

```
X = np.array([[0, 1], [1, 0]])
Y = np.array([[0, -1j], [1j, 0]])
Z = np.array([[1, 0], [0, -1]])

def plot_bloch_path(bloch_points):
    fig = plt.figure(figsize=(6, 6))
    ax = fig.add_subplot(111, projection='3d')


    u, v = np.mgrid[0:2*np.pi:60j, 0:np.pi:30j]
    x = np.cos(u)*np.sin(v)
    y = np.sin(u)*np.sin(v)
    z = np.cos(v)
    ax.plot_surface(x, y, z, color='lightgray', alpha=0.15,
        linewidth=0)

    ax.plot(bloch_points[:, 0], bloch_points[:, 1],
        bloch_points[:, 2],
            color='red', linewidth=2)

    start = bloch_points[0]

    ax.quiver(0, 0, 0, 1.2, 0, 0, color='k',
        arrow_length_ratio=0.1)
    ax.quiver(0, 0, 0, 0, 1.2, 0, color='k',
        arrow_length_ratio=0.1)
    ax.quiver(0, 0, 0, 0, 0, 1.2, color='k',
        arrow_length_ratio=0.1)
    ax.text(1.3, 0, 0, r'$\hat{n}_x$', fontsize=12)
    ax.text(0, 1.3, 0, r'$\hat{n}_y$', fontsize=12)
    ax.text(0, 0, 1.3, r'$\hat{n}_z$', fontsize=12)

    ax.view_init(elev=25, azim=45)
    ax.set_box_aspect([1, 1, 1])
    ax.set_title('Bloch Sphere Path (rotated view)')
    plt.show()

def simulate_berry_phase(h_vec, T=2*np.pi, Nt=1000,
                         plot_overlap=True, bloch=True):

    dt = T / Nt
    E0 = -1.0

```

```
    psi0 = np.array([1, -1]) / np.sqrt(2)

    psi = psi0.copy()


    time_steps = [0.0]
    overlaps_amp = [1.0]
    total_phases = [0.0]
    berry_phases = [0.0]
    dynamical_phases_acc = [0.0]
    bloch_points = [h_vec(0.0)]

    for n in range(Nt):
        t_current = (n + 1) * dt
        lmbd_for_H = n * dt / T
        lmbd_current = t_current / T

        h = h_vec(lmbd_for_H)
        H = h[0]*X + h[1]*Y + h[2]*Z
        U = expm(-1j * H * dt)
        psi = U @ psi

        O = np.vdot(psi0, psi)

        current_total_phase = -np.angle(O)
        current_dyn_phase = E0 * t_current
        current_berry_phase = current_total_phase -
            current_dyn_phase

        time_steps.append(t_current)
        overlaps_amp.append(np.abs(O))
        total_phases.append(current_total_phase)
        berry_phases.append(current_berry_phase)
        dynamical_phases_acc.append(current_dyn_phase)
        bloch_points.append(h_vec(lmbd_current))

    time_steps = np.array(time_steps)
    overlaps_amp = np.array(overlaps_amp)
    total_phases = np.array(total_phases)
    berry_phases = np.array(berry_phases)
    dynamical_phases_acc = np.array(dynamical_phases_acc)
    bloch_points = np.array(bloch_points)

    if plot_overlap:

        plt.figure(figsize=(6, 3))
        plt.plot(time_steps, overlaps_amp, color='royalblue',
            label='$|O(t)|$')
        plt.axhline(y=1.0, color='gray', linestyle='--',
            linewidth=1,
                    label='Initial state ($|O|=1$)')
        plt.title('$|O(t)|$ (Overlap Amplitude)')
        plt.xlabel('Time (t)')
```

```
        plt.ylabel('Amplitude')
        plt.legend(loc='lower right')
        plt.ylim(-0.1, 1.1)
        plt.tight_layout()
        plt.show()

        plt.figure(figsize=(6, 3))

        plt.plot(time_steps, np.unwrap(total_phases),
                 label=r'$\theta_{total}(t)$ (Total Phase)')
        plt.plot(time_steps, np.unwrap(berry_phases),
                 label=r'$\theta_{Berry}(t)$ (Berry Phase)')
        plt.plot(time_steps, dynamical_phases_acc,
                 label=r'Dynamical Phase ($\int E_0 dt$)',
                 linestyle='--', color='gray')

        plt.title('Phase Evolution')
        plt.xlabel('Time (t)')
        plt.ylabel('Phase (rad)')
        plt.legend()
        plt.tight_layout()
        plt.show()


    if bloch:
        plot_bloch_path(bloch_points)

    final_berry_phase = total_phases[-1] % (2 * np.pi)
    print(f'Final Berry phase (mod 2pi) = {final_berry_phase
        :.3f} rad')
```

Output 1: Berry phase for $\vec{h}_0(\lambda)$.

```
Final Berry phase = 6.283 rad
```

Output 2: Berry phase for $\vec{h}_1(\lambda)$.

```
Final Berry phase = 2.455rad
```

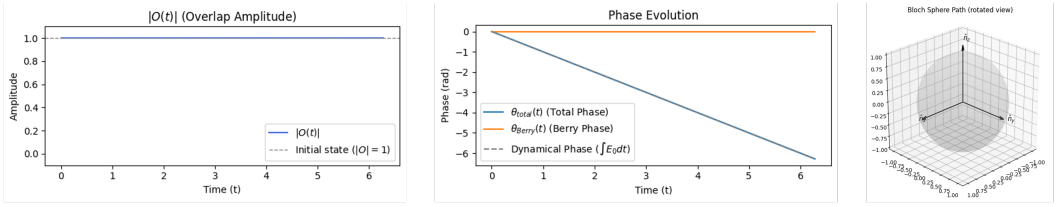
Output 3: Berry phase for $\vec{h}_2(\lambda)$.

```
Final Berry phase = 4.974 rad
```

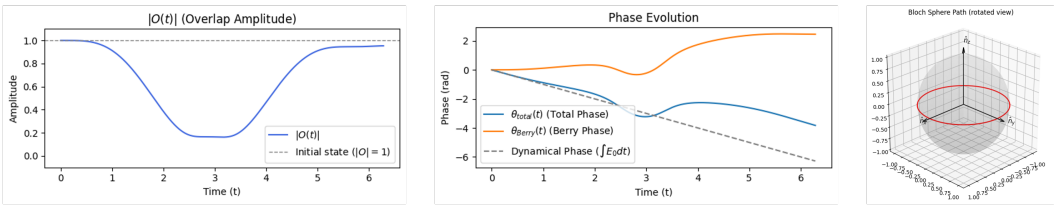
Output 4: Berry phase for $\vec{h}_3(\lambda)$.

```
Final Berry phase = 4.065 rad
```

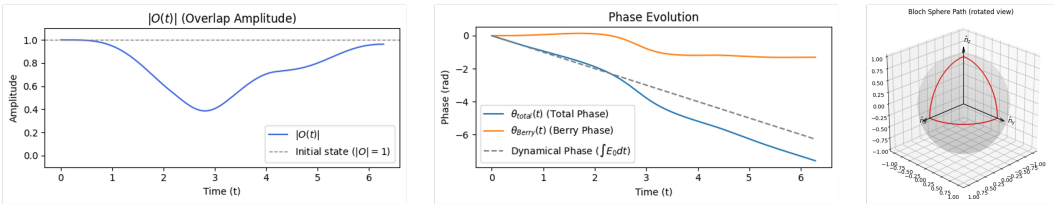
In the first case, since the vector does not change over time, there is no variation in $|O(t)|$, and the Berry phase is found to be 2π, which is equivalent to zero when considering periodicity.



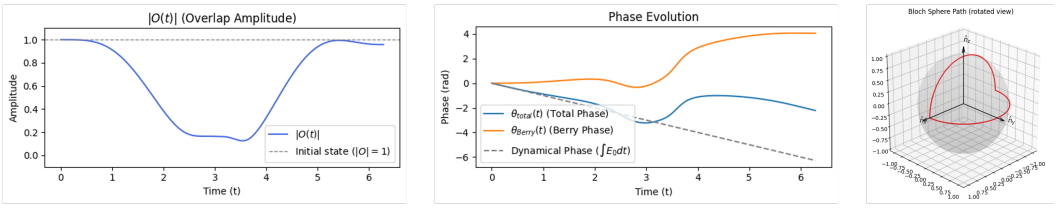
(a)



(b)



(c)



(d)

Figure 2: Simulation plot graph for each trajectory.

However, for the remaining three cases, it can be seen that $|O(t)|$ at the final timestep slightly differs from its initial value $|O(0)|$. Furthermore, the fact that the Berry phase has a nonzero value indicates that, in addition to the dynamical phase arising from time evolution, a geometric phase (Berry phase) also exists, which originates from the trajectory taken in the parameter space.

Part 2: How to measure Berry phase with quantum computers?

Question 2.1

The given quantum state in this problem is:

$$|\psi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{i\theta}|1\rangle\right) \tag{35}$$

To construct a quantum circuit that generates this state, we start from the initial qubit state $|0\rangle$ and use a Hadamard gate to prepare the intermediate superposition state. After applying the Hadamard gate, the state becomes:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle \tag{36}$$

To add a phase to $|1\rangle$, we apply the $R_z(\theta)$ gate. When $R_z(\theta)$ is applied to $|+\rangle$, we obtain:

$$R_z(\theta)|+\rangle = \frac{1}{\sqrt{2}}\left(e^{-i\theta/2}|0\rangle + e^{i\theta/2}|1\rangle\right) \tag{37}$$

Multiplying the entire state by a global phase factor $e^{i\theta/2}$ yields the same physical state as in the problem statement. Although this global phase is physically unobservable, the resulting qubit state is equivalent to the target state.

Hence, the target state can be represented by applying H followed by $R_z(\theta)$ to the initial state. However, since the R_z gate performs a rotation around the z-axis, it affects only the phase, not the measurement probabilities.

To observe how the parameter θ affects the measurement results, we can apply another Hadamard gate, as it transforms phase differences into measurable probability amplitudes in the computational basis.

Applying H to $|\psi\rangle$ gives:

$$H|\psi\rangle = \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\ e^{i\theta}\end{pmatrix} = \frac{1}{2}\begin{pmatrix}1+e^{i\theta}\\ 1-e^{i\theta}\end{pmatrix} \tag{38}$$

The measurement probabilities are then determined by the squared magnitudes of the amplitudes:

$$P(0) = \left|\frac{1+e^{i\theta}}{2}\right|^2 = \cos^2\left(\frac{\theta}{2}\right) \tag{39}$$

$$P(1) = \left|\frac{1-e^{i\theta}}{2}\right|^2 = \sin^2\left(\frac{\theta}{2}\right) \tag{40}$$

Thus, by measuring these probabilities, we can infer the value of θ.

The overall process can be implemented using the Qiskit library. The code used for the test is shown in Listing 3, and the circuit structure is illustrated in Fig. 3. The test was performed with $\theta = \pi/3$. In this case, the theoretical measurement probabilities are $P(0) = 0.75$ and $P(1) = 0.25$.

Listing 3: Code that measures the phase of a single qubit system.

```
theta = np.pi / 3

qc = QuantumCircuit(1, 1)
qc.h(0)
qc.rz(theta, 0)
qc.h(0)
qc.measure(0, 0)

qc.draw('mpl')
```

```
10
11 sim = AerSimulator()
12 result = sim.run(qc, shots=100000).result()
13 counts = result.get_counts()
14
15 print('Measurement counts:', counts)
```

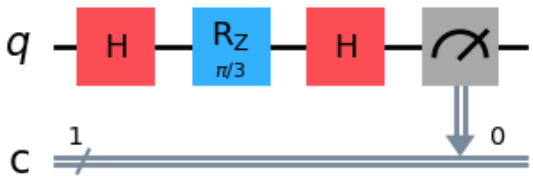


Figure 3: Quantum circuit implementing the measurement of phase θ.

The measurement result (Output 5) closely matches the theoretical probabilities, confirming that the observed results are consistent with the expected outcome.

Output 5: Single-qubit system measurement result.

```
Measurement counts: {'1': 24914, '0': 75086}
```

Question 2.2

In this problem, our goal is to measure the total phase accumulated by the reference qubit after it undergoes time evolution under a time-independent Hamiltonian.

In this setup, the reference qubit serves as an ancilla that generates interference. To create this interference, a Hadamard gate is applied to the reference qubit. Let the system qubit be denoted as $|E\rangle$, and the initial combined state $|\Psi_0\rangle$ is expressed as:

$$|\Psi_0\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle |E\rangle + |1\rangle |E\rangle \right) \tag{41}$$

To encode the phase accumulated over time into the basis state $|1\rangle$ of the reference qubit, a Controlled-U operation is applied. This operation ensures that the system undergoes time evolution only when the reference qubit is in state $|1\rangle$. After performing the Controlled-U operation, the combined state $|\Psi_1\rangle$ becomes:

$$|\Psi_1\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle |E\rangle + e^{-iHT} |1\rangle |E\rangle \right) \tag{42}$$

Thus, a relative phase difference is introduced between $|0\rangle$ and $|1\rangle$, where the phase difference is $\Delta\theta = HT$.

Next, as in the previous problem, applying a Hadamard gate allows the two paths to interfere, thereby converting the phase difference into measurable probabilities. Applying a Hadamard gate to the reference qubit gives:

$$H \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} e^{iHT} \\ 1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} e^{iHT} \\ 1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 + e^{iHT} \\ 1 - e^{iHT} \end{pmatrix} \tag{43}$$

Therefore, the final state of the reference qubit $|\psi_{sub}\rangle$ is:

$$|\psi_{sub}\rangle = \frac{1}{2} \left[(1 + e^{iHT}) |0\rangle + (1 - e^{iHT}) |1\rangle \right] \tag{44}$$

The measurement probabilities appear as squared magnitudes of these amplitudes:

$$P(0) = \left| \frac{1 + e^{-iHT}}{2} \right|^2 = \cos^2 \left(\frac{HT}{2} \right) \tag{45}$$

$$P(1) = \left| \frac{1 - e^{-iHT}}{2} \right|^2 = \sin^2 \left(\frac{HT}{2} \right) \tag{46}$$

Hence, the phase HT can be inferred through the measurement of the reference qubit.

The corresponding Qiskit implementation is shown in Listing 4, and the quantum circuit used in this case is illustrated in Fig. 4. In this code, the Hamiltonian H is assumed to be the Pauli-X Hamiltonian with eigenvalues of ± 1. The system qubit is initially prepared in the $|+\rangle$ state, and the time evolution operator is given by $U = e^{-iXT}$. Since $R_x(\phi) = e^{-iX\phi/2}$, this operator can be equivalently represented as a rotation gate $R_x(2T)$. In the code, $T = \pi/4$ is assumed; therefore, the theoretical value of $P(0)$ is $\cos^2(\pi/8) = 0.8536$, and that of $P(1)$ is $\sin^2(\pi/8) = 0.1464$.

Listing 4: Code of total phase measurement circuit for a time-independent Hamiltonian.

```
E = 1.0
T = np.pi / 4
theta = E * T

qc = QuantumCircuit(2, 1)

qc.h(1)

qc.h(0)
qc.crx(2*theta, 0, 1)
qc.h(0)
qc.measure(0, 0)

qc.draw('mpl')

sim = AerSimulator()
result = sim.run(qc, shots=100000).result()
counts = result.get_counts()

print('Measurement counts:', counts)
```

The measurement result is the same as output 6, and it can be seen that the measurement result close to the theoretical probability appears.

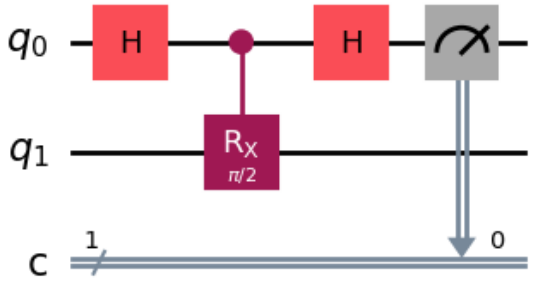


Figure 4: Quantum circuit implementing the total-phase measurement under a time-independent Hamiltonian.

Output 6: Two-qubit (ancilla–system) total-phase measurement result under a time-independent Hamiltonian.

```
Measurement counts: {'0': 85278, '1': 14722}
```

Question 2.3

In this problem, we consider the case where the Bloch vector $\vec{h}(t)$ changes over time. The overall structure of the circuit is almost identical to that in Question 2.2, but because the vector changes with time, multiple Controlled-Unitary gates corresponding to each time step are added. As the number of time steps increases (i.e., as the time interval becomes smaller), the accuracy improves, but the circuit complexity also increases exponentially. Therefore, it is important to maintain a proper balance.

The trajectory follows the four paths shown in Question 1.3. The initial state corresponds to the ground state of $\vec{h}(0) = (1, 0, 0)$, denoted by $|-\rangle$. Since all other processes are identical, the circuit can be constructed using Listing 5.

Listing 5: Time-varying Hamiltonian-based Berry phase measurement quantum circuit generation code.

```
def create_berry_phase_circuit(h_vec_func, T=2*np.pi, Nt=1000):

    dt = T / Nt
    qc = QuantumCircuit(2, 1)

    qc.x(1)
    qc.h(1)
    qc.h(0)

    for n in range(Nt):
        lmbd = n / Nt
        h = h_vec_func(lmbd)
```

```
        H_n = h[0]*X + h[1]*Y + h[2]*Z
        U_n_matrix = expm(-1j * H_n * dt)

        U_gate = UnitaryGate(U_n_matrix, label=f'U_{n}')
        controlled_U_gate = U_gate.control(1)

        qc.append(controlled_U_gate, [0, 1])

    qc.h(0)
    qc.measure(0, 0)

    return qc
```

The measured results allow us to determine the phase. From Eq. 45, we obtain:

$$\sqrt{P(0)} = \cos\left(\frac{\theta_{\text{total}}}{2}\right) \tag{47}$$

$$\cos^{-1}\left(\sqrt{P(0)}\right) = \frac{\theta_{\text{total}}}{2} \tag{48}$$

$$\theta_{\text{total}} = 2 \cdot \cos^{-1}\left(\sqrt{P(0)}\right) \tag{49}$$

In this setup, since the total time for all four cases is set as an integer multiple of 2π, the dynamical phase θ_{dyn} does not contribute, resulting in $\theta_{\text{total}} = \theta_{\text{berry}}$.

The code used to perform the measurement and calculate the Berry phase is shown in Listing 6.

Listing 6: Code to calculate Berry phase from quantum circuit measurement results.

```
def simulate_berry_phase_circuit(h_vec, T=2*np.pi, Nt=1000,
    shots=100000):
    berry_qc = create_berry_phase_circuit(h_vec, T, Nt)
    berry_qc_decomposed = berry_qc.decompose()

    sim = AerSimulator()
    result = sim.run(berry_qc_decomposed, shots=shots).result
        ()
    counts = result.get_counts()

    total_shots = counts.get('0', 0) + counts.get('1', 0)
    p0 = counts.get('0', 0) / total_shots
    theta_total = 2 * np.arccos(np.sqrt(p0))

    final_berry_phase = theta_total % (2 * np.pi)
    print('Measurement counts:', counts)
    print(f'Calculated Berry phase from circuit (mod 2pi) = {
        final_berry_phase:.3f} rad')
```

By performing the simulation for the four trajectories, the results are obtained as shown in Outputs 7–10.

Output 7: Berry phase for $\vec{h}_0(\lambda)$ (In quantum circuit).

```
Measurement counts: {'0': 100000}
Calculated Berry phase from circuit (mod 2pi) = 0.000 rad
```

Output 8: Berry phase for $\vec{h}_1(\lambda)$ (In quantum circuit).

```
Measurement counts: {'0': 13061, '1': 86939}
Calculated Berry phase from circuit (mod 2pi) = 2.402 rad
```

Output 9: Berry phase for $\vec{h}_2(\lambda)$ (In quantum circuit).

```
Measurement counts: {'0': 62222, '1': 37778}
Calculated Berry phase from circuit (mod 2pi) = 1.324 rad
```

Output 10: Berry phase for $\vec{h}_3(\lambda)$ (In quantum circuit).

```
Measurement counts: {'0': 21100, '1': 78900}
Calculated Berry phase from circuit (mod 2pi) = 2.187 rad
```

The results show that the calculated values are nearly identical to those obtained in Outputs 1–4. This confirms that the values are consistent, even though the overall phase sign cannot be distinguished due to the nature of the formula. Additionally, for shorter paths or different angular periodicities, as in Outputs 9–10 (which appear different from Outputs 3–4), the computed values are also verified to be correct.

Part 3: How does Berry phase appear in topological phases?

Question 3.1

The given tight-binding Hamiltonian is expressed as follows [1] :

$$H_{\mathrm{SSH}} = -\left[(J+\delta J)\sum_{j=0(\mathrm{mod}\ 2)} + (J-\delta J)\sum_{j=1(\mathrm{mod}\ 2)}\right] c_{j+1}^{\dagger} c_j + \mathrm{h.c.} \tag{50}$$

Here, $J \pm \delta J$ represent alternating hopping amplitudes, indicating the coupling strengths between neighboring lattice sites. Each unit cell consists of two sublattices. If we index the unit cells by j, the annihilation operators of the two sublattice sites within the same cell are defined as c_{2j} and c_{2j+1}. Although the Hamiltonian in real space couples multiple sites, applying a Fourier transformation simplifies it into a more compact form.

The Fourier transformation provided in the problem is given by:

$$\begin{pmatrix} c_{2j} \\ c_{2j+1} \end{pmatrix} = \frac{1}{\sqrt{N_{\mathrm{uc}}}} \sum_k \begin{pmatrix} c_{k0} \\ c_{k1} \end{pmatrix} e^{ik\cdot r_{2j}} \tag{51}$$

where N_{uc} is the number of unit cells, $k = 2\pi n/N_{\mathrm{uc}}$, and $n \in [-N_{\mathrm{uc}}/2,\ N_{\mathrm{uc}}/2]$.

Next, we apply the Fourier transform to each term of the Hamiltonian. We separately evaluate the terms corresponding to even and odd j. First, consider the term associated with even j:

$$-(J+\delta J)\sum_{j=0(\mathrm{mod}\ 2)} c_{j+1}^{\dagger}c_j \tag{52}$$

Since j is even, i.e., $j = 2j'$, this term can be rewritten as:

$$-(J+\delta J)\sum_{j} c_{2j+1}^{\dagger}c_{2j} \tag{53}$$

Separating the Fourier components from Eq. 51, we have:

$$c_{2j} = \frac{1}{\sqrt{N_{\mathrm{uc}}}}\sum_{k} c_{k0}e^{ikr_{2j}}, \quad c_{2j+1} = \frac{1}{\sqrt{N_{\mathrm{uc}}}}\sum_{k'} c_{k'1}e^{ik'r_{2j}} \tag{54}$$

Substituting these into the original Hamiltonian term gives:

$$-(J+\delta J)\sum_{j} c_{2j+1}^{\dagger}c_{2j} = -(J+\delta J)\sum_{j}\left(\frac{1}{\sqrt{N_{\mathrm{uc}}}}\sum_{k'} c_{k'1}^{\dagger}e^{-ik'r_{2j}}\right)\left(\frac{1}{\sqrt{N_{\mathrm{uc}}}}\sum_{k} c_{k0}e^{ikr_{2j}}\right) \tag{55}$$

$$= -(J+\delta J)\frac{1}{N_{\mathrm{uc}}}\sum_{j}\sum_{k,k'} c_{k'1}^{\dagger}c_{k0}e^{i(k-k')r_{2j}} \tag{56}$$

Here, $k - k' = 2\pi(n-n')/N$, and the factor $2\pi/N$ indicates that periodicity arises from the phase difference between wave numbers. When $k \neq k'$, the terms $e^{i(k-k')r_{2j}}$ average out to zero due to destructive interference. However, for $k = k'$, we have $e^{i(k-k')r_{2j}} = 1$. Thus, only the terms with $k = k'$ remain, yielding:

$$-(J+\delta J)\sum_{k} c_{k1}^{\dagger}c_{k0} \tag{57}$$

Similarly, the term corresponding to odd j can be derived analogously. It should be noted that the even-site term c_{2j} and the odd-site term c_{2j+1} belong to the same unit cell, whereas c_{2j+1} and c_{2j+2} belong to neighboring cells. Hence, a spatial phase factor e^{-ik} is introduced, and the corresponding expression becomes:

$$-(J-\delta J)\sum_{k} c_{k0}^{\dagger}c_{k1}e^{-ik} \tag{58}$$

Combining the transformed terms, the Hamiltonian of the SSH model in momentum space can be written as:

$$H = -(J+\delta J)\sum_{k} c_{k1}^{\dagger}c_{k0} - (J-\delta J)\sum_{k} c_{k0}^{\dagger}c_{k1}e^{-ik} + \mathrm{h.c.} \tag{59}$$

Here, h.c. denotes the Hermitian conjugate of the preceding terms. Expanding the full expression, we obtain:

$$H = -(J+\delta J)\sum_k (c_{k1}^\dagger c_{k0} + c_{k0}^\dagger c_{k1}) - (J-\delta J)\sum_k (c_{k0}^\dagger c_{k1} e^{-ik} + c_{k1}^\dagger c_{k0} e^{ik}) \quad (60)$$

We can express this in matrix form by defining $c_k = (c_{k0}, c_{k1})^T$, yielding:

$$\begin{aligned} H_k &= \begin{pmatrix} 0 & -(J+\delta J) - (J-\delta J)e^{-ik} \\ -(J+\delta J) - (J-\delta J)e^{ik} & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & q^*(k) \\ q(k) & 0 \end{pmatrix} \end{aligned} \quad (61)$$

Since $e^{ik} = \cos k + i \sin k$ and $e^{-ik} = \cos k - i \sin k$, we can substitute these expressions to obtain h_x and h_y as follows:

$$h_x = -(J+\delta J) - (J-\delta J)\cos k, \qquad h_y = -(J-\delta J)\sin k.$$

Using these transformed vector components and the Pauli matrices, the Hamiltonian $\mathcal{H}_k$ can be written as

$$\mathcal{H}_k = h_x \sigma_x + h_y \sigma_y,$$

or equivalently,

$$\mathcal{H}_k = (h_x, h_y, 0) \cdot \vec{\sigma} = \vec{h}_k \cdot \vec{\sigma}.$$

By calculating the eigenvalues of the Hamiltonian and plotting the results, the energy band structure can be obtained. The code that performs this sequence of steps is shown in Listing 7, and the visualization of the energy bands as a function of δJ is presented in Fig. 5.

Listing 7: Calculation code for energy band structure of SSH model.

```
J = 1
deltaJ_values = [-0.5, 0.0, 0.5]
k_vals = np.linspace(-np.pi, np.pi, 400)

for deltaJ in deltaJ_values:

    h_x = -(J + deltaJ) - (J - deltaJ) * np.cos(k_vals)
    h_y = -(J - deltaJ) * np.sin(k_vals)

    E = np.sqrt(h_x**2 + h_y**2)

    plt.figure(figsize=(4, 5))
    plt.plot(k_vals, E, label='Conduction band (+)')
    plt.plot(k_vals, -E, label='Valence band (-)')

    plt.title(f'SSH Model Band Structure (J=1, delta_J={deltaJ
        })', fontsize=12)
    plt.xlabel('Momentum k')
    plt.ylabel('Energy E(k)')
    plt.xlim(-np.pi, np.pi)
```

```
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

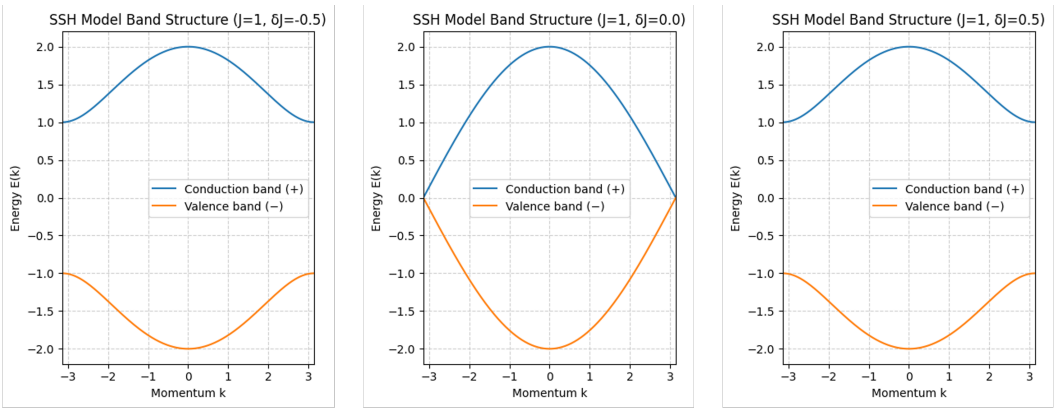


Figure 5: Energy band structure of the SSH model.

The blue line in the upper part of the graph represents the conduction band, while the orange line in the lower part represents the valence band. Although the graphs for $\delta J = -0.5$ and $\delta J = +0.5$ appear identical in shape, they correspond to two states with opposite topological phases. When $\delta J = -0.5$, the inter-cell coupling is stronger, causing the electrons to be localized along the cell boundaries, which corresponds to a *topological insulator* state. In contrast, when $\delta J = +0.5$, the intra-cell coupling becomes stronger, and the electrons are localized within the unit cell, representing a *trivial insulator* state. At $\delta J = 0$, the band gap closes, indicating a *topological phase transition point* between the topological and trivial insulator phases. Therefore, even though the band shapes are identical, the sign of δJ determines the bonding configuration and the resulting topological nature of the system.

Question 3.2

In the previous question, we confirmed that the band structures at $\delta J = -0.5$ and $\delta J = 0.5$ are identical. Thus, the band shape alone cannot determine the presence of a Berry phase. In this problem, we explicitly calculate the Berry phase of the SSH model to verify that the phases of the two ground states are indeed different.

In this case, since the Bloch vector is not a unit vector, the ground-state energy E_{0k} depends on k. Therefore, even if we define the period as 2π, the total phase does not necessarily cancel out dynamically. To address this, we employ the **half-time-reversal method**, in which the system is adiabatically evolved while $k(t)$ changes over time, and then evolved backward to cancel the dynamical phase, leaving only the Berry phase.

Listing 8 simulates this entire procedure for $\delta J \in [-1, 1]$. Instead of a continuous integration, the Berry phase is approximated by summing up the phase changes

between neighboring k-points.

Listing 8: SSH Model Half-Time-Reversal Code for calculating Berry phase.

```

def Hk(k: float, J: float = 1.0, dJ: float = 0.0):
    hx = -(J + dJ) - (J - dJ) * np.cos(k)
    hy = -(J - dJ) * np.sin(k)
    hz = 0.0
    return hx * X + hy * Y + hz * Z

def unitary_from_H(H, dt):
    trH2 = np.trace(H @ H).real
    hnorm = math.sqrt(max(trH2 / 2.0, 0.0))
    if hnorm < 1e-12:
        return np.eye(2, dtype=complex)
    c = math.cos(hnorm * dt)
    s = math.sin(hnorm * dt) / hnorm
    return c * np.eye(2, dtype=complex) - 1j * s * H

def ground_state(H):
    evals, evecs = np.linalg.eigh(H)
    idx = evals.argmin()
    return evecs[:, idx], evals[idx].real

def evolve_path_unitary(k_path, T_half, steps, J, dJ, backward
   =False):
    dt = T_half / steps
    U = np.eye(2, dtype=complex)
    for k in k_path:
        U_step = unitary_from_H(Hk(k, J, dJ), dt)
        if backward:
            U = U_step.conj().T @ U
        else:
            U = U_step @ U
    return U

def berry_phase_half_time_reversal(J=1.0, dJ=0.0, T=200*np.pi,
    Nt=2000):
    steps = Nt // 2
    T_half = T / 2
    t1 = np.linspace(0, T_half, steps, endpoint=False)
    k1 = np.linspace(-math.pi, 0, steps + 1)
    k2 = np.linspace(0, math.pi, steps + 1)


    psi_i, _ = ground_state(Hk(-math.pi, J, dJ))
    U1 = evolve_path_unitary(k1, T_half, steps, J, dJ)
    U2 = evolve_path_unitary(k2, T_half, steps, J, dJ,
       backward=True)
    U_total = U2 @ U1
    psi_f = U_total @ psi_i

```

```
    O = np.vdot(psi_i, psi_f)
    theta = cmath.phase(O)
    overlap_mag = abs(O)

    P0 = np.array([[1,0],[0,0]], dtype=complex)
    P1 = np.array([[0,0],[0,1]], dtype=complex)
    CU = np.kron(P0, np.eye(2, dtype=complex)) + np.kron(P1,
        U_total)
    plus = np.array([1,1], dtype=complex) / np.sqrt(2)
    psi0 = np.kron(plus, psi_i)
    psi_after = CU @ psi0

    Hgate = (1/np.sqrt(2)) * np.array([[1,1],[1,-1]], dtype=
        complex)
    U_measX = np.kron(Hgate, np.eye(2, dtype=complex))
    psi_mx = U_measX @ psi_after
    amp0 = psi_mx[0:2]
    ex = 2*np.vdot(amp0, amp0).real - 1

    Sdg = np.array([[1,0],[0,-1j]], dtype=complex)
    U_measY = np.kron(Hgate @ Sdg, np.eye(2, dtype=complex))
    psi_my = U_measY @ psi_after
    amp0y = psi_my[0:2]
    ey = 2*np.vdot(amp0y, amp0y).real - 1

    theta_qc = math.atan2(ey, ex)

    def wrap(a): return (a + 2*np.pi) % (2*np.pi)
    return wrap(theta), wrap(theta_qc), overlap_mag


def run_sweep_and_plot(J=1.0, dJ_min=-1.0, dJ_max=1.0, num=41,
    T=200*np.pi, Nt=2000):
    dJs = np.linspace(dJ_min, dJ_max, num)
    thetas, thetas_qc, overlaps = [], [], []

    for dJ in dJs:
        th, th_qc, ov = berry_phase_half_time_reversal(J, dJ,
            T, Nt)
        thetas.append(th)
        thetas_qc.append(th_qc)
        overlaps.append(ov)

    plt.figure(figsize=(14,6))
    plt.plot(dJs, thetas, label='Overlap', linewidth=1.8)
    plt.plot(dJs, thetas_qc, '--', label='Ancilla', linewidth
        =1.5)

    plt.axhline(0, color='gray', linestyle=':', linewidth=1.2)
    plt.axhline(2*np.pi, color='gray', linestyle='--',
        linewidth=1.2)
    plt.axhline(np.pi, color='gray', linestyle='--', linewidth
        =1.2)
```

```
    plt.text(dJ_max*0.85, 0.15, 'theta', color='gray')
    plt.text(dJ_max*0.85, 2*np.pi+0.15, '2pi', color='gray')
    plt.text(dJ_max*0.85, np.pi+0.15, 'pi', color='gray')

    plt.xlabel('delta J')
    plt.ylabel('Berry phase theta (rad)')
    plt.title('SSH Berry phase')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(14,6))
    plt.plot(dJs, overlaps, color='tab:orange')
    plt.xlabel('delta J')
    plt.ylabel('|<psi_i|psi_f>|')
    plt.title('Overlap magnitude (Adiabaticity check)')
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()
```

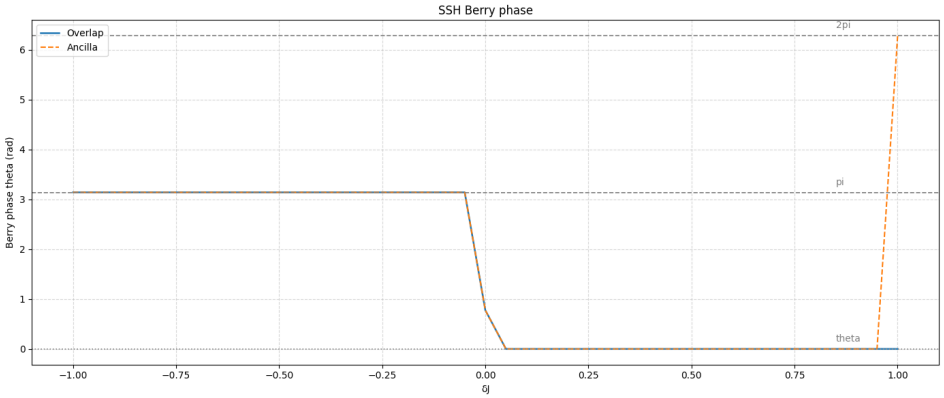


Figure 6: Berry phase of SSH model as a function of δJ.

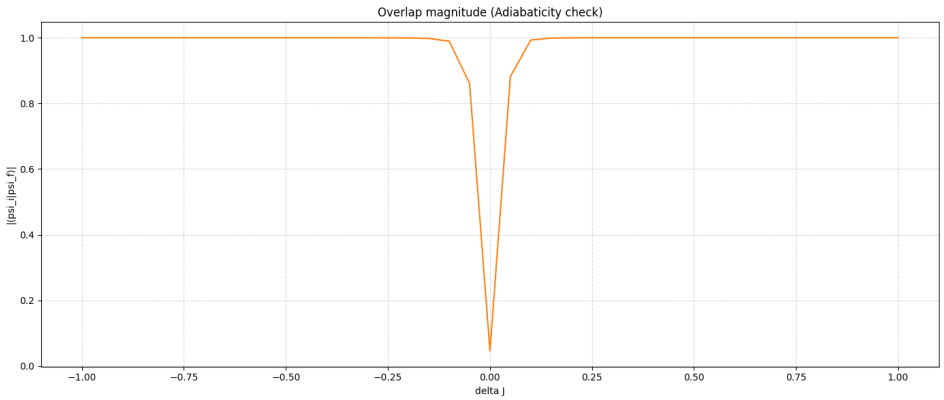


Figure 7: Adiabaticity check via ground-state overlap magnitude.

When the code is executed, the results appear as shown in Fig. 6 and Fig. 7.

Fig. 6 shows the variation of the Berry phase of the ground state as δJ is swept from -1 to 1. The blue line represents the theoretical value, while the orange line shows the result obtained from simulation. The Berry phase abruptly changes from π to 0 near $\delta J = 0$, which marks the topological phase transition.

Fig. 7 presents the overlap magnitude between the initial and final ground states, $|\langle\psi_i|\psi_f\rangle|$, serving as a measure of adiabaticity. The overlap remains nearly unity except near $\delta J = 0$, where it drops sharply, indicating that the energy gap closes at the transition point.

Together, Fig. 6 and 7 provide numerical evidence consistent with the analytical results of Fig. 5, confirming that the SSH model undergoes a topological phase transition at $\delta J = 0$, where the Berry phase changes discontinuously.

Question 3.3

Up to this point, all procedures have been analyzed based on fermions. However, to implement this on a quantum computer, we must use qubits instead of fermions. Therefore, fermionic operators must be transformed into qubit operators.

The creation operator $c_j^\dagger$ adds a fermion, while the annihilation operator c_j removes a fermion. On a qubit, this action corresponds to flipping the spin, which is implemented using the X and Y gates. Thus, we have:

$$c_j^\dagger \sim \frac{X_j - iY_j}{2}, \qquad c_j \sim \frac{X_j + iY_j}{2}. \tag{62}$$

However, it is not appropriate to use these forms directly. Fermions exhibit the property that a sign changes when two fermions are exchanged. That is,

$$c_1 c_2 = -c_2 c_1. \tag{63}$$

On qubits, though, exchanging operators does not change the sign. Therefore, an additional sign correction must be included. In this process, we multiply a string of Z operators, $\prod_{k<j} Z_k$. This ensures that, when constructing c_j, all previous fermions contribute a factor of Z, reflecting the sign change due to fermionic parity. Thus, the complete operators $c_j^\dagger$ and c_j can be written as:

$$c_j^\dagger = \left(\prod_{k<j} Z_k\right) \frac{X_j - iY_j}{2}, \qquad c_j = \left(\prod_{k<j} Z_k\right) \frac{X_j + iY_j}{2}. \tag{64}$$

To express the full Hamiltonian, we also need the number operator n_j in addition to $c_j^\dagger$ and c_j. The operator n_j indicates whether a fermion exists at site j. It is expressed as the product of $c_j^\dagger$ and c_j: if an electron exists, the value is 1; if not, it is 0. Using the transformation derived above, n_j can also be written in terms of gates. The transformation proceeds as follows:

$$
\begin{aligned}
n_j &= c_j^\dagger c_j \\
&= \left(\prod_{k<j} Z_k\right)\left(\frac{X_j - iY_j}{2}\right)\left(\prod_{k<j} Z_k\right)\left(\frac{X_j + iY_j}{2}\right) \\
&= \frac{1}{4}\left(\prod_{k<j} Z_k\right)^2 (X_j - iY_j)(X_j + iY_j). \qquad (65)
\end{aligned}
$$

Since $Z_k^2 = I$, we have

$$\left(\prod_{k<j} Z_k\right)^2 = I. \qquad (66)$$

Therefore,

$$\frac{1}{4}(X_j - iY_j)(X_j + iY_j) = \frac{1}{4}(X_j^2 + iX_jY_j - iY_jX_j + Y_j^2). \qquad (67)$$

Since X_j^2 and Y_j^2 are equal to I, and $iX_jY_j - iY_jX_j = -2Z_j$, the final transformed expression for n_j becomes

$$n_j = \frac{2I - 2Z_j}{4} = \frac{I - Z_j}{2}. \qquad (68)$$

Through this process, we see that the fermionic occupations $\{0, 1\}$ are directly mapped to the qubit basis states $\{|0\rangle, |1\rangle\}$.

Next, we verify that the transformed operators satisfy the true fermionic anti-commutation relations. First, we consider the case where the indices are the same, i.e., $j = k$.

$$\{c_j, c_j^\dagger\} = c_j c_j^\dagger + c_j^\dagger c_j. \qquad (69)$$

Using the previously derived Pauli-operator expressions, $c_j c_j^\dagger$ becomes

$$
\begin{aligned}
c_j c_j^\dagger &= \left(\prod_{k<j} Z_k\right)^2 \left(\frac{X_j + iY_j}{2}\right)\left(\frac{X_j - iY_j}{2}\right) \\
&= \frac{1}{4}(X_j^2 + Y_j^2 + i[X_j, Y_j]) \\
&= \frac{1}{4}(I + I + i(2iZ_j)) \\
&= \frac{1}{2}(I - Z_j). \qquad (70)
\end{aligned}
$$

Similarly, for $c_j^\dagger c_j$ we obtain

$$c_j^\dagger c_j = \frac{1}{2}(I + Z_j). \qquad (71)$$

Adding these two results gives I, so

$$\{c_j, c_j^\dagger\} = I = 1. \tag{72}$$

Therefore, this matches the correct fermionic anticommutation relation.

Next, we consider the case $j \neq k$. Rewriting c_j and $c_k^\dagger$ gives

$$c_j = \left(\prod_{p<j} Z_p\right) \frac{X_j + iY_j}{2}, \qquad c_k^\dagger = \left(\prod_{q<k} Z_q\right) \frac{X_k - iY_k}{2}. \tag{73}$$

Now, assuming $j < k$, we can rewrite the product $\prod_{q<k} Z_q$ as:

$$\prod_{q<k} Z_q = \left(\prod_{p<j} Z_p\right) Z_j Z_{j+1} \cdots Z_{k-1}. \tag{74}$$

Therefore, $c_j c_k^\dagger$ can be written as:

$$\begin{aligned} c_j c_k^\dagger &= \left[\left(\prod_{p<j} Z_p\right) \frac{X_j + iY_j}{2}\right] \left[\left(\prod_{p<j} Z_p\right) (Z_j Z_{j+1} \cdots Z_{k-1}) \frac{X_k - iY_k}{2}\right] \\ &= \frac{1}{4} (Z_j Z_{j+1} \cdots Z_{k-1}) (X_j + iY_j)(X_k - iY_k). \end{aligned} \tag{75}$$

Similarly, for $c_k^\dagger c_j$ we have:

$$c_k^\dagger c_j = \frac{1}{4} (Z_j Z_{j+1} \cdots Z_{k-1}) (X_k - iY_k)(X_j + iY_j). \tag{76}$$

Since the order of the operators in $c_j c_k^\dagger$ and $c_k^\dagger c_j$ is reversed, they acquire a minus sign. Thus,

$$c_j c_k^\dagger = -c_k^\dagger c_j, \tag{77}$$

and therefore

$$\{c_j, c_k^\dagger\} = c_j c_k^\dagger + c_k^\dagger c_j = 0. \tag{78}$$

This confirms that, through the Jordan–Wigner transformation, the qubit-gate expressions for fermionic operators indeed satisfy the correct fermionic anticommutation relations even after the mapping.

Question 3.4

In this step, we apply the Jordan–Wigner transformation obtained previously to the full SSH model. Writing the SSH Hamiltonian with explicit hopping terms gives:

$$H_{\text{SSH}} = -(J + \delta J) \sum_{j=0 \,(\text{mod}\, 2)} c_{j+1}^\dagger c_j - (J - \delta J) \sum_{j=1 \,(\text{mod}\, 2)} c_{j+1}^\dagger c_j + V \sum_j n_j n_{j+1} + \text{h.c.} \tag{79}$$

Here, the term $V \sum_j n_j n_{j+1}$ represents the additional repulsive interaction between electrons.

We begin by applying the Jordan–Wigner transformation to $c_{j+1}^\dagger c_j$:

$$\begin{aligned} c_{j+1}^\dagger c_j &= \left(\prod_{k<j+1} Z_k \right) \frac{X_{j+1} - iY_{j+1}}{2} \left(\prod_{k<j} Z_k \right) \frac{X_j + iY_j}{2} \\ &= \frac{1}{4} Z_j (X_{j+1} - iY_{j+1})(X_j + iY_j) \end{aligned} \tag{80}$$

Including the hermitian conjugate:

$$c_{j+1}^\dagger c_j + c_j^\dagger c_{j+1} = \frac{1}{2}(X_{j+1}X_j + Y_{j+1}Y_j) \tag{81}$$

Next, we transform the density–density interaction:

$$n_j n_{j+1} = \frac{1 - Z_j}{2} \cdot \frac{1 - Z_{j+1}}{2} \tag{82}$$

Since the model assumes periodic boundary conditions (PBC), the term involving $c_0^\dagger c_{N-1}$ must also be included. The operators c_0 and c_{N-1} are expressed as:

$$\begin{aligned} c_0 &= \frac{X_0 + iY_0}{2}, \\ c_{N-1} &= \left(\prod_{k<N-1} Z_k \right) \frac{X_{N-1} + iY_{N-1}}{2} \end{aligned} \tag{83}$$

In the SSH model, the total number of sites N is always even because the chain consists of pairs forming unit cells. Thus, the periodic boundary hopping between the last site $(N-1)$ and the first site (0) corresponds to inter-cell hopping with coefficient $-(J - \delta J)$.

Finally, the fully transformed qubit Hamiltonian is:

$$\begin{aligned} H_{\text{SSH}} = &-\frac{J + \delta J}{2} \sum_{j=0 \,(\mathrm{mod}\, 2)} (X_{j+1}X_j + Y_{j+1}Y_j) \\ &- \frac{J - \delta J}{2} \sum_{j=1 \,(\mathrm{mod}\, 2)} (X_{j+1}X_j + Y_{j+1}Y_j) \\ &+ \frac{V}{4} \sum_{j}^{N-2} (1 - Z_j - Z_{j+1} + Z_j Z_{j+1}) \\ &+ \Bigg(\frac{J - \delta J}{2} \Bigg(\prod_{k<N-1} Z_k \Bigg) (X_0 X_{N-1} + Y_0 Y_{N-1}) \\ &+ \frac{V}{4} (1 - Z_{N-1} - Z_0 + Z_{N-1} Z_0) \Bigg) \end{aligned} \tag{84}$$

Question 3.5

The PBC (periodic boundary condition) considered in this problem can be viewed intuitively as a "circular ring." When the ring is rotated by one full cycle, from a quantum-mechanical point of view, a phase may be generated. That is,

$$\psi(x+L) = e^{i\phi}\psi(x) \tag{85}$$

Thus, the periodicity of the ring under PBC can be implemented through this phase.

Therefore, the part of the Hamiltonian H_{PBC} that connects the electrons at the two ends of the chain must include the phase factor ϕ.

Applying this phase to the term $c_0^\dagger c_{N-1} + c_{N-1}^\dagger c_0$:

$$e^{-i\phi}c_0^\dagger c_{N-1} + e^{i\phi}c_{N-1}^\dagger c_0 \tag{86}$$

Since $e^{i\phi} = \cos\phi + i\sin\phi$ and $e^{-i\phi} = \cos\phi - i\sin\phi$, we have

$$e^{-i\phi}c_0^\dagger c_{N-1} + e^{i\phi}c_{N-1}^\dagger c_0 = \frac{1}{2}\left(\prod_{k<N-1} Z_k\right)\left[\cos\phi(X_0X_{N-1} + Y_0Y_{N-1}) + \sin\phi(X_0Y_{N-1} - Y_0X_{N-1})\right] \tag{87}$$

Therefore, the full PBC Hamiltonian including the phase θ is:

$$\begin{aligned} H_{\text{PBC}}(\theta) = {} & \frac{J-\delta J}{2}\left(\prod_{k<N-1} Z_k\right)\left[\cos\theta(X_0X_{N-1} + Y_0Y_{N-1}) + \sin\theta(X_0Y_{N-1} - Y_0X_{N-1})\right] \\ & + \frac{V}{4}(1 - Z_{N-1} - Z_0 + Z_{N-1}Z_0) \end{aligned} \tag{88}$$

Finally, combining the PBC term with the full SSH Hamiltonian, the total Hamiltonian with phase θ becomes:

$$\begin{aligned} H_{\text{SSH}}(\theta) = {} & -\frac{J+\delta J}{2}\sum_{j=0\,(\text{mod}\,2)}(X_{j+1}X_j + Y_{j+1}Y_j) \\ & - \frac{J-\delta J}{2}\sum_{j=1\,(\text{mod}\,2)}(X_{j+1}X_j + Y_{j+1}Y_j) \\ & + \frac{V}{4}\sum_{j}^{N-2}(1 - Z_j - Z_{j+1} + Z_jZ_{j+1}) \\ & + \Bigg(\frac{J-\delta J}{2}\left(\prod_{k<N-1} Z_k\right)\left[\cos\theta(X_0X_{N-1} + Y_0Y_{N-1}) + \sin\theta(X_0Y_{N-1} - Y_0X_{N-1})\right] \\ & + \frac{V}{4}(1 - Z_{N-1} - Z_0 + Z_{N-1}Z_0)\Bigg) \end{aligned} \tag{89}$$

Question 3.6

In this stage, we assume that there is no interaction term, i.e., $V = 0$. Therefore, the SSH model used here is simplified as follows:

$$\begin{aligned} H_{\mathrm{SSH}}(\theta) = &-\frac{J+\delta J}{2} \sum_{j=0 \,(\mathrm{mod}\, 2)} (X_{j+1}X_j + Y_{j+1}Y_j) \\ &- \frac{J-\delta J}{2} \sum_{j=1 \,(\mathrm{mod}\, 2)} (X_{j+1}X_j + Y_{j+1}Y_j) \\ &+ \left(\frac{J-\delta J}{2} \left(\prod_{k<N-1} Z_k \right) \right) \left[\cos\theta \left(X_0 X_{N-1} + Y_0 Y_{N-1} \right) + \sin\theta \left(X_0 Y_{N-1} - Y_0 X_{N-1} \right) \right] \end{aligned} \tag{90}$$

Now, we vary δJ in the outer loop from -1 to 1, and for each value of δJ, we slowly change the inner-loop parameter θ from 0 to 2π to compute the corresponding Berry phase. In this simulation, the Hamiltonian is projected onto the half-filling subspace with $N = 8$ sites and 4 particles. The simulation code for calculating the Berry phase as a function of δJ follows the structure shown in Listing 9, and the results are presented in Fig. 8 and Fig. 9.

Listing 9: Code for calculating the bary phase of the gate-transformed Hamiltonian circuit.

```
def H_theta(theta, J, deltaJ, V, N, X, Y, Z):
    H = np.zeros((2**N, 2**N), dtype=complex)

    for j in range(N - 1):
        coeff = -(J + deltaJ) / 2.0 if j % 2 == 0 else -(J -
            deltaJ) / 2.0
        H += coeff * (X[j+1] @ X[j] + Y[j+1] @ Y[j])

    Z_string = np.eye(2**N, dtype=complex)
    for k in range(0, N - 1):
        Z_string = Z[k] @ Z_string

    coeff_pbc = -(J - deltaJ) / 2.0
    term_cos = np.cos(theta) * (X[0] @ X[N-1] + Y[0] @ Y[N-1])
    term_sin = np.sin(theta) * (X[0] @ Y[N-1] - Y[0] @ X[N-1])

    H += coeff_pbc * (Z_string @ (term_cos + term_sin))

    for j in range(N):
        j_next = (j + 1) % N
        term = np.eye(2**N) - Z[j] - Z[j_next] + Z[j] @ Z[
            j_next]
        H += (V / 4.0) * term

    return H

```

```
def berry_phase_twist_simulation(J=1.0, dJ=0.0, V=0.0, N=8, steps=41):
    X, Y, Z = get_pauli_ops(N)
    basis_idx = get_half_filling_indices(N)

    thetas = np.linspace(0, 2*np.pi, steps, endpoint=False)
    ground_states = []

    for th in thetas:
        H_full = H_theta(th, J, dJ, V, N, X, Y, Z)
        H_sub = project_hamiltonian(H_full, basis_idx)

        vals, vecs = eigh(H_sub)
        ground_states.append(vecs[:, 0])

    overlap_prod = 1.0 + 0.0j
    for i in range(steps):
        psi_curr = ground_states[i]
        psi_next = ground_states[(i + 1) % steps]
        overlap_prod *= np.vdot(psi_curr, psi_next)

    theta_berry = -np.angle(overlap_prod)
    if theta_berry < -0.001:
        theta_berry += 2*np.pi

    overlap_mag = np.abs(overlap_prod)

    return theta_berry, theta_berry, overlap_mag


def run_sweep_and_plot(J=1.0, V=0.0, dJ_min=-1.0, dJ_max=1.0, num=51):
    N_sites = 8
    n_steps = 100

    dJs = np.linspace(dJ_min, dJ_max, num)
    thetas, thetas_qc, overlaps = [], [], []

    for dJ in dJs:
        th, th_qc, ov = berry_phase_twist_simulation(J, dJ, V,
            N_sites, n_steps)
        thetas.append(th)
        thetas_qc.append(th_qc)
        overlaps.append(ov)

    plt.figure(figsize=(14, 6))
    plt.plot(dJs, thetas, '-', label='Overlap (Calculated)',
       linewidth=1.8)

    plt.axhline(0, color='gray', linestyle=':', linewidth=1.2)
    plt.axhline(2 * np.pi, color='gray', linestyle='--',
       linewidth=1.2)
    plt.axhline(np.pi, color='gray', linestyle='--', linewidth
```

```
        =1.2)

    plt.xlabel('$\delta J$')
    plt.ylabel('Berry phase $\Theta$ (rad)')
    plt.title(f'SSH Berry phase (Interaction V={V})')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(14, 6))
    plt.plot(dJs, overlaps, '-', color='tab:orange')
    plt.xlabel('$\delta J$')
    plt.ylabel('|Product of Overlaps|')
    plt.title('Adiabaticity check (Magnitude)')
    plt.ylim(0.0, 1.1)
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()

run_sweep_and_plot()
```

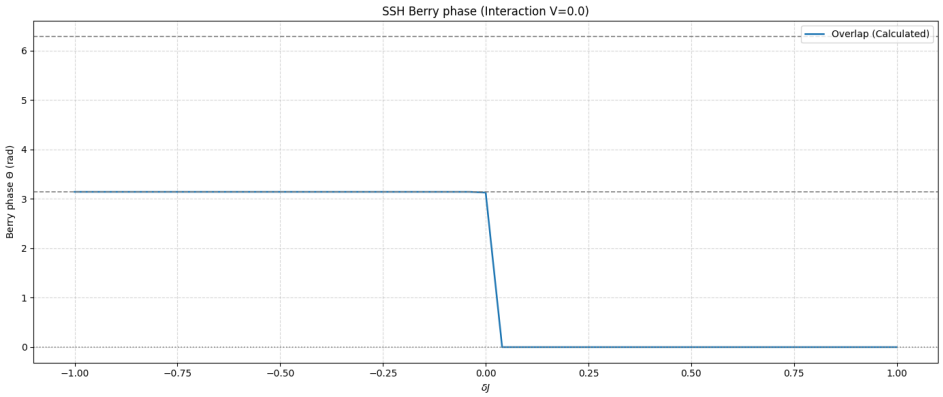


Figure 8: Berry phase of gate-transformed SSH hamiltonian.

We previously confirmed in Section 3.2 that the Berry phase approaches π for $\delta J < 0$, while it approaches 0 for $\delta J > 0$, and that it becomes unstable near $\delta J = 0$ due to the closing of the energy gap. Through Fig. 8 and Fig. 9, we successfully verify that this behavior also holds for the Hamiltonian transformed into quantum gates.

Question 3.7

In the previous step, we examined how the Berry phase changes with variations in δJ when $V = 0$. Next, keeping $\delta J = -0.5$, we will vary the electron-electron interaction strength V and observe how the Berry Phase changes.

The overall structure is similar to the previous implementation, but it differs in that the value of V is varied from 0 to 20. The implementation code is shown in

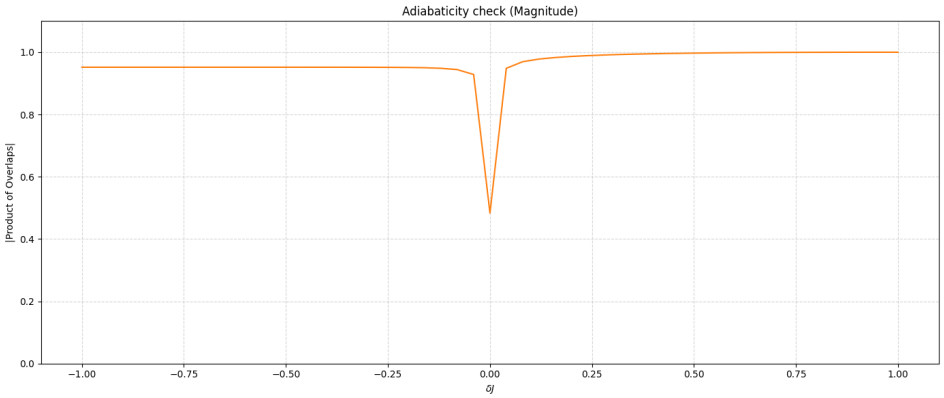


Figure 9: Overlap magnitude for gate-transformed SSH hamiltonian.

Listing 10.

Listing 10: Code to calculate SSH model Berry phase according to boundary twist.

```
def build_ssh_hamiltonian(theta, J, deltaJ, V, N, X, Y, Z):
    H = np.zeros((2**N, 2**N), dtype=complex)

    for j in range(N - 1):
        coeff = -(J + deltaJ)/2.0 if j % 2 == 0 else -(J -
            deltaJ)/2.0
        H += coeff * (X[j+1] @ X[j] + Y[j+1] @ Y[j])

    Z_string = np.eye(2**N, dtype=complex)
    for k in range(N - 1):
        Z_string = Z[k] @ Z_string

    dim = 2**N

    Z_string = np.eye(dim, dtype=complex)
    for k in range(N - 1):
        Z_string = Z[k] @ Z_string


    boundary_term = (
        X[N-1] @ Z_string @ X[0] +
        Y[N-1] @ Z_string @ Y[0]
    )
    coeff_pbc = -(J - deltaJ) / 2.0
    term_cos = np.cos(theta) * (X[0] @ X[N-1] + Y[0] @ Y[N-1])
    term_sin = np.sin(theta) * (X[0] @ Y[N-1] - Y[0] @ X[N-1])

    H += coeff_pbc * (Z_string @ (term_cos + term_sin))


    for j in range(N):
        j_next = (j + 1) % N
        term = np.eye(2**N) - Z[j] - Z[j_next] + Z[j] @ Z[
```

```
            j_next]
        H += (V / 4.0) * term

    return H


def analyze_ssh_phase(J, deltaJ, V, N, steps=100):
    X, Y, Z = get_pauli_ops(N)
    basis_idx = get_half_filling_indices(N)

    H0 = build_ssh_hamiltonian(0, J, deltaJ, V, N, X, Y, Z)
    H0_sub = project_hamiltonian(H0, basis_idx)
    vals, _ = eigh(H0_sub)
    gap = vals[1] - vals[0]

    thetas = np.linspace(0, 2*np.pi, steps, endpoint=False)
    ground_states = []
    for th in thetas:
        H = build_ssh_hamiltonian(th, J, deltaJ, V, N, X, Y, Z
            )
        H_sub = project_hamiltonian(H, basis_idx)
        _, vecs = eigh(H_sub)
        ground_states.append(vecs[:, 0])

    overlap_prod = 1.0 + 0.0j
    for i in range(steps):
        overlap_prod *= np.vdot(ground_states[i],
            ground_states[(i + 1) % steps])

    berry_phase = np.angle(overlap_prod)
    if berry_phase < -0.001: berry_phase += 2*np.pi

    return berry_phase, gap


def run_full_analysis():
    N = 8
    J = 1.0
    deltaJ = -0.5
    V_list = np.linspace(0, 20, 41)

    phases = []
    gaps = []

    for V in V_list:
        bp, gap = analyze_ssh_phase(J, deltaJ, V, N)
        phases.append(bp)
        gaps.append(gap)

    fig, ax1 = plt.subplots(figsize=(12, 6))

    color = 'tab:purple'
    ax1.set_xlabel('Interaction Strength V')
```

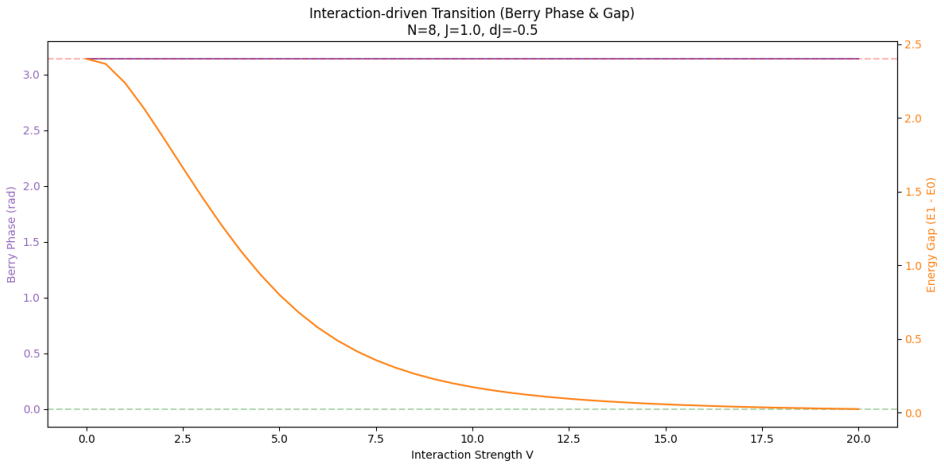


Figure 10: Variation of Berry phase and energy gap with interaction intensity V.

```
    ax1.set_ylabel('Berry Phase (rad)', color=color)
    ax1.plot(V_list, phases, '-', color=color, label='Berry
        Phase')
    ax1.tick_params(axis='y', labelcolor=color)
    ax1.axhline(np.pi, color='r', linestyle='--', alpha=0.3,
        label='Topological')
    ax1.axhline(0, color='g', linestyle='--', alpha=0.3, label
        ='Trivial')

    ax2 = ax1.twinx()
    color = 'tab:orange'
    ax2.set_ylabel('Energy Gap (E1 - E0)', color=color)
    ax2.plot(V_list, gaps, '-', color=color, label='Energy Gap
        ')
    ax2.tick_params(axis='y', labelcolor=color)

    plt.title(f'Interaction-driven Transition (Berry Phase &
        Gap)\nN={N}, J={J}, dJ={deltaJ}')
    fig.tight_layout()
    plt.show()

run_full_analysis()
```

Fig.10 shows the results of running the code in Listing 9, illustrating the Berry phase and energy gap as a function of V. According to the simulation results, as the interaction strength V increases, the energy gap tends to approach zero, suggesting the possibility of inducing a phase transition. However, since the energy gap does not fully reach zero, a clear phase transition was not observed. Therefore, even as V increases, the Berry phase remains close to π. This result can be interpreted as the topological robustness associated with $\delta J = -0.5$, and furthermore, due to finite-size effects in the small system of $N = 8$, gap closing does not occur completely.

Question 3.8

In the final step, we carry out Sections 3.6 and 3.7 simultaneously. Specifically, we vary both δJ and V at the same time and visually examine the resulting changes in the energy gap ΔE. For intuitive understanding, we generated a heatmap visualization, and the corresponding implementation is provided in Listing 11.

Listing 11: Code for calculating the energy gap for the $(\delta J, V)$ parameter space in the interacting SSH model.

```
def build_ssh_hamiltonian(J, deltaJ, V, N, X, Y, Z):

    dim = 2**N
    H = np.zeros((dim, dim), dtype=complex)

    for j in range(N - 1):
        if j % 2 == 0:
            t = J + deltaJ
        else:
            t = J - deltaJ

        coeff = -(t) / 2.0
        H += coeff * (X[j+1] @ X[j] + Y[j+1] @ Y[j])

    coeff_pbc = -(J - deltaJ) / 2.0

    Z_string = np.eye(dim, dtype=complex)
    for k in range(N - 1):
        Z_string = Z[k] @ Z_string


    boundary_term = (
        X[N-1] @ Z_string @ X[0] +
        Y[N-1] @ Z_string @ Y[0]
    )

    H += coeff_pbc * boundary_term

    for j in range(N):
        j_next = (j + 1) % N
        term = (
            np.eye(dim, dtype=complex)
            - Z[j]
            - Z[j_next]
            + Z[j] @ Z[j_next]
        )
        H += (V / 4.0) * term

    return H


def compute_gap(J, deltaJ, V, N, X, Y, Z, basis_idx):
    H = build_ssh_hamiltonian(J, deltaJ, V, N, X, Y, Z)
```

```
    H_sub = project_hamiltonian(H, basis_idx)

    vals = np.linalg.eigvalsh(H_sub)
    vals = np.sort(vals)
    return vals[1] - vals[0]


def plot_phase_diagram():
    N = 8
    J = 1.0
    dJ_res = 41
    V_res = 41

    dJs = np.linspace(-1.0, 1.0, dJ_res)
    Vs = np.linspace(0.0, 20.0, V_res)

    gaps = np.zeros((V_res, dJ_res))

    X, Y, Z = get_pauli_ops(N)
    basis_idx = get_half_filling_indices(N)

    for i, Vval in enumerate(Vs):
        for j, dJval in enumerate(dJs):
            gaps[i, j] = compute_gap(J, dJval, Vval, N, X, Y,
                Z, basis_idx)

    plt.figure(figsize=(10, 7))
    extent = [dJs.min(), dJs.max(), Vs.min(), Vs.max()]

    plt.imshow(gaps, origin='lower', extent=extent, aspect='
        auto', cmap='inferno')
    plt.colorbar(label=r'Energy Gap $\Delta E = E_1 - E_0$')
    plt.xlabel(r'$\delta J$')
    plt.ylabel(r'$V$')
    plt.title('SSH Energy Gap Diagram\n$N=8, J=1$')

    plt.tight_layout()
    plt.show()


plot_phase_diagram()
```

Fig.11 shows the results obtained from executing Listing 10. The diagram exhibits left–right symmetry with respect to the transformation from $-\delta J$ to δJ. This symmetry arises because the SSH model possesses a unitary symmetry that flips the sign of δJ through unit cell rearrangement or sublattice exchange. Thus, the preservation of this symmetry confirms that the Hamiltonian has been implemented correctly.

Furthermore, for a fixed value of δJ, the energy gap ΔE tends to decrease as V increases. This behavior can be understood by noting that, in the strong-V regime, many surrounding states acquire nearly identical energies as long as they satisfy the condition of minimizing adjacent occupancies. In other words, strong interactions

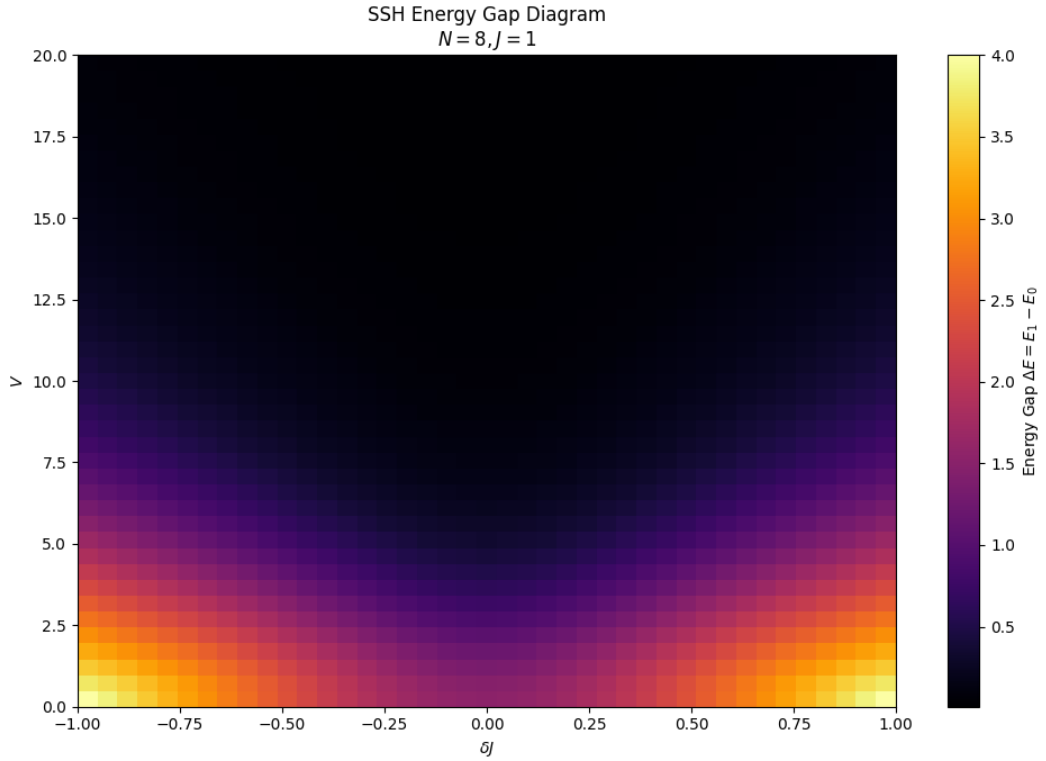


Figure 11: Energy gap heatmap over $(\delta J, V)$ in the interacting SSH model.

generate an almost degenerate manifold of states, resulting in a reduction of the difference $E_1 - E_0$.

Conclusion

In this project, we explored the concept of the Berry phase and its role in various quantum systems. In Part 1, we examined how the Berry phase emerges under adiabatic evolution, derived its analytical form, and computed the geometric phase for several trajectories on the Bloch sphere. In Part 2, we extended this understanding to quantum circuits, demonstrating how interference and ancilla-assisted measurement allow us to extract global phase information that is otherwise inaccessible through standard measurements. In Part 3, we applied these ideas to the SSH Hamiltonian by mapping it into a qubit-based representation and computing the energy spectrum while varying model parameters. This enabled us to investigate interaction effects and identify how the energy gap evolves across different regimes, providing insight into topological phase transitions.

Overall, our results highlight the significance of the Berry phase as a fundamental quantity that connects quantum geometry, quantum computation, and topological matter. Developing the ability to simulate and measure such phases on quantum devices is expected to play an increasingly important role in future quantum technologies.

References

[1] Tony Jin, Paola Ruggiero, and Thierry Giamarchi. Bosonization of the interacting su-schrieffer-heeger model. *Phys. Rev. B*, 107, 2023.