# Assignment 3: Quantum statistical mechanics and thermalization

Instructor: Lesik Motrunich
TA: Liam O'Brien

Ph 121C: Computational Physics Lab, Spring 2024
California Institute of Technology
Due: 4 pm Tuesday, May 14, 2024

## 1 Introduction to quantum thermalization

In this assignment we broaden our focus, previously restricted to ground and low-energy states, to include the entire spectrum as well as the time evolution of quantum states under a manybody Hamiltonian. While time evolution as governed by the Schrödinger equation is perhaps not too difficult for a single particle (i.e., quantum mechanics), in an interacting system of many constituents the situation is evidently much more complicated. Composite classical systems are described on macroscopic scales by statistical mechanics, but we do not have a formal connection between this theory and the microscopic physics. For this reason many questions remain open: one example is the problem of deriving thermodynamic irreversibility - the general tendency to approach equilibrium-from microscopic equations which are symmetric in time. However, it seems essential for the validity of statistical mechanics that a system explores its entire microscopic phase space (subject to constraints like energy conservation) given enough time, a property called ergodicity. The natural mechanism for ergodicity is chaotic dynamics, in which similar initial microscopic configurations diverge exponentially quickly in time under the nonlinear equations of motion governing the constituents.

The analogous quantum situation is the time evolution of an isolated pure state of a manybody system, referred to as a quench. As the equation of motion for a closed quantum system is linear, chaos of the type encountered classically is prohibited. In addition, the unitarity of the time evolution operator $\exp(-itH)$ requires the dynamics to be strictly reversible, as the initial state of a quantum system can be reconstructed from the state at later times by the application of the backwards-time operator with $t \mapsto -t$. It was paradoxical, then, when in the early 1990s both Srednicki and Deutsch proposed that an isolated quantum system under time evolution does indeed generically approach a type of equilibrium thermal state, independent of both the properties of the initial state and the fine details of the Hamiltonian, similar to the case in classical statistical mechanics. Moreover, this was predicted to occur in isolated states in the absence of coupling to a thermal bath or any kind of reservoir.

Subsequent numerical simulations as well as analytic work and experimental evidence support the claim, now known as the eigenstate thermalization hypothesis (ETH), in a variety of quantum systems. As we will see, there are multiple ways to understand the phenomenon of quantum thermalization, including by examining those cases where ETH is violated. In the following we will explore both ETH and non-ETH dynamics using exact diagonalization of spin systems similar to the quantum Ising model with which you are already familiar.

# 2 Eigenstate thermalization hypothesis

## 2.1 Quantum chaos

As described above, the linearity of Schrödinger's equation prohibits the exponential divergence of trajectories in phase space which is the signature of chaotic dynamics. In fact, we cannot really make a direct comparison because the uncertainty principle prohibits localizing quantum states in phase space; however, we note that time evolution actually preserves overlaps:

$$\langle \phi(t) \mid \psi(t) \rangle = \left\langle \phi(0) \left| U^\dagger(t) U(t) \right| \psi(0) \right\rangle = \langle \phi(0) \mid \psi(0) \rangle \tag{1}$$

The modern understanding of quantum chaos is due to work from the 1960s and 1970s when Wigner and others were studying the spectra of large atomic nuclei. These energy eigenvalues, while experimentally measurable, had too complicated a structure to describe with an analytic form; instead, Wigner's insight was that if one focuses on a small energy window the Hamiltonian looks like a matrix with random entries when represented in a generic basis. Thus, the eigenvalues of quantum Hamiltonians at finite energy density should be described by the statistical properties of random matrices, subject to some physical symmetry constraints. This topic is known as random matrix theory (RMT), and is quite successful in describing energy eigenvalues, particularly via "level statistics." RMT also predicts that the eigenvectors of such Hamiltonians look nearly stochastic, and moreover turn out to be very sensitive to small perturbations; thus a slightly perturbed Hamiltonian written in the unperturbed eigenbasis already looks again like a random matrix. Because the eigenvectors of the Hamiltonian are the stationary states of the system, this is understood as the quantum counterpart to classical exponentially diverging trajectories.

## 2.2 Dynamical ETH

The canonical picture of classical thermalization involves a small (finite) system in contact with an infinitely large reservoir. Over time, the smaller system will come to equilibrium with the reservoir through the transfer of energy by various microscopic processes, which are not themselves important. Initially, the smaller system may have certain distinct characteristics, such as a particular profile for its energy density or magnetization, but these are lost as the system reaches equilibrium and takes on the average properties of the reservoir. That is, after a sufficiently long time, the only relevant quantity is in fact the temperature $\tau$ of the reservoir, and the system explores its microstates according to the Boltzmann distribution, which associates a probability $p(\varepsilon) \propto e^{-\beta\varepsilon}$ with a microstate of energy $\varepsilon$ (here, $\beta = 1/\tau$ ). In particular, the reverse process is exceedingly unlikely: a system in equilibrium is not expected to reconstruct its initial conditions before a time which is doubly exponential in the system size. This is known as the Poincaré recurrence time.

One formulation of ETH is the claim that the thermalization behavior described above applies also to small subsystems of an isolated quantum system in the thermodynamic limit undergoing time evolution. We will make this statement more precise by considering the expectation values of local observables, supported on a few sites, in the time-evolving quantum state. Let $|\psi(0)\rangle = \sum_n c_n |n\rangle$
be some initial state along with its decomposition in the eigenbasis of $H, \{|n\rangle\}_{n=1,\ldots,2^N}$. In this basis, the time-evolved state is

$$|\psi(t)\rangle = \sum_n c_n e^{-i\varepsilon_n t} |n\rangle \tag{2}$$

where $\varepsilon_n$ is the energy eigenvalue corresponding to $|n\rangle$. Now suppose that we measure some local observable $O$ in the state as a function of time. Its expectation value is given by

$$O(t) \equiv \langle\psi(t)|O|\psi(t)\rangle = \sum_{m,n} c_m^* c_n e^{-i(\varepsilon_n-\varepsilon_m)t}\langle m|O|n\rangle \tag{3}$$

$$= \sum_n |c_n|^2 O_{nn} + \sum_{m,n\neq m} c_m^* c_n e^{-i(\varepsilon_n-\varepsilon_m)t} O_{mn} \tag{4}$$

Taking a long-time average of the observable, and absent degeneracies in the spectrum, the offdiagonal terms in (4) will be eliminated by dephasing; thus, the time average will approach the time-independent value given by the first sum, which is based only on the initial conditions. In this sense, equilibration is actually a generic feature of quantum quenches.

The statement of ETH is much stronger, however: not only do local observables $O$ equilibrate in time, but they approach a specific thermal value. Recall that we may appeal to RMT when focused on a narrow energy window, say of width $\delta$; this corresponds to an initial state $|\psi(0)\rangle$ with only small energy fluctuations. Then the predicted thermal value of the operator is in fact its average in the eigenstates within the energy window:

$$O_{\mathrm{MC}} = \frac{1}{N_{\varepsilon\pm\delta}} \sum_{\{n:|\varepsilon_n-\varepsilon|<\delta\}} O_{nn} \tag{5}$$

where $N_{\varepsilon\pm\delta}$ is the number of eigenstates $|n\rangle$ with energy $|\varepsilon_n - \varepsilon| < \delta$. This is the average used to measure quantities in the microcanonical ensemble in statistical mechanics. This picture can be extended to make a broader claim about the spectrum in the case that the only conserved (local) quantity in the system is the energy. Then the prediction of ETH is the measurement corresponding to a "Gibbs state:"

$$O_\beta = \frac{1}{Z_\beta} \mathrm{tr}\left[e^{-\beta H} O\right] \tag{6}$$

where the partition function $Z_\beta = \mathrm{tr}\left[e^{-\beta H}\right]$. This is the average in the canonical ensemble at temperature $\tau = 1/\beta$. To be more precise, strong ETH is the claim that for any local observable and any initial state, in the thermodynamic limit the late-time value approaches the form (6). Another formulation, weak ETH, also asserts the above but only for typical-or "almost all"local observables and initial states.

By probing local observables time-evolving under an interacting Hamiltonian from initial states with low entanglement, we emulate the classical picture of thermalization between a small system (the support of the operator) and a reservoir (the remainder of the sites). However, in doing so we actually provide the resolution of the paradox between late-time equilibration and the reversibility of unitary dynamics. It turns out that the information about the initial state is not lost in the equilibrium state, but instead is "smeared out" from its initial configuration and becomes encoded nonlocally in the time-evolved state. Operators acting on extensively many sites, then, will not appear thermal even after local observables have equilibrated, and reversing the dynamics restores the initial state by again localizing the information specific to the initial condition. The apparent directionality of the quench is based on the choice of an initial state having low entanglement, which contains very little information in extensive operators and as we saw in the previous assignment is atypical in the full Hilbert space.

## 2.3 Eigenstate ETH

The arguments of the previous section lead to a paradox: if the long-time behavior of an observable $O$ is given by the diagonal terms of (4), which depend on the initial state, how can it approach the universal form given by (5)? The resolution is that dynamical ETH is equivalent to thermalization of the eigenstates - hence the name - which is to say that expectation values of operators in an eigenstate at energy $\varepsilon_n$ are essentially determined

by the thermal value (6) at the corresponding $\beta(\varepsilon_n)$. The precise statement for the matrix elements of local operators in the energy eigenbasis is

$$O_{mn} = O_\beta(\varepsilon_n)\,\delta_{mn} + e^{-S(\bar{\varepsilon})/2} f_O(\varepsilon_m, \varepsilon_n)\, R_{mn} \tag{7}$$

where $\bar{\varepsilon}$ is the average of $\varepsilon_m$ and $\varepsilon_n$, $S$ the thermodynamic entropy, and $f_O(\cdot,\cdot)$ is some smooth function modulating a random variable $R_{mn}$. It is essential that $O_\beta(\varepsilon_n)$ is a smooth function, and for (5) to hold, measurements in nearby eigenstates should look nearly the same: $O_{n+1,n+1} - O_{n,n}$ is exponentially small in system size in the middle of the spectrum, where energies $\varepsilon_n$ and $\varepsilon_{n+1}$ are also exponentially close.

However, the arguments for thermalization do not apply near the edges of the spectrum: these distinguished energy regions generally behave quite differently from the rest of the eigenstates, which has been a recurring theme in the previous assignments. In particular, the density of states is lower at the band edges, and so heuristically it is more difficult for eigenstates to "hybridize" with one another as the Hamiltonian is perturbed. Recall the values of the ground state fidelity you found in Assignment 1 for the gapped phases of the Ising model; these should have been nearly 1 and thus represent a clear violation of the prediction of RMT applied to the ground state.

# 3 Failure to thermalize

One very interesting question is, what are the limits of the picture described above? That is, do all systems exhibit thermalization, or are there some necessary assumptions that can be violated? The avoidance of thermalization has important real-world implications, for example for quantum information devices that need to be shielded from decoherence. It turns out that in principle certain Hamiltonians can indeed avoid thermalization. These fall into two classes which superficially are quite different, but turn out to have a similar underlying property which prevent initial states from reaching equilibrium, namely that the evolution is strongly constrained by conservation laws.

## 3.1 Integrable quantum systems

An integrable system is one respecting an extensive number of conserved quantities, or local integrals of motion. These conservation laws restrict the dynamics to such an extent that, once identified, they allow exact solution of the model. For example, the one-dimensional quantum Ising model in a transverse field is integrable. This is seen by mapping to spinless fermions through the JordanWigner transformation, which results in a free theory of particles hopping on a lattice. The model is solved independently for each of the fermionic momentum states; then, generating multi-particle states simply corresponds to setting the occupation of each of the single-particle orbitals.

In such a system thermalization is avoided due to the non-interaction of the constituents. More generally, two-body interacting models can also be integrable if the interaction terms satisfy a particular condition known as the Yang-Baxter equation. The addition of a longitudinal field $\sum_j \sigma_j^x$ to the Ising model breaks its integrability, introducing an interaction term (between the fermions) of the type that eliminates the conservation of single-particle states.

## 3.2 Many-body localization

Recently, a great deal of study has been devoted to a newer mechanism of avoided thermalization, which is known as many-body localization (MBL). MBL is a phenomenon in which fixed randomness (so-called "quenched disorder," not to be confused with the previous usage of quench) in some of the terms in the Hamiltonian causes all eigenstates of an interacting system to become localized, with a transition from delocalized to localized eigenstates occurring at some finite strength of the disorder.

Since Anderson, it has been known that disorder in a non-interacting system will result in localization of the single-particle states, which may either occur for arbitrarily weak disorder or else above some transition value. However, it is conventionally expected that allowing interactions leads to resonances which delocalize the many-body states, presumably following ETH as in the quantum integrable case when interactions are introduced. Instead, numerical evidence indicates that above a localization transition, strong disorder localizes all eigenstates. This transition to MBL is qualitatively different from the quantum phase transitions we studied previously, as it occurs throughout the entire spectrum.

In order to understand MBL, one can refer to "l-bits," which are emergent local integrals of motion (for weak disorder they can be thought of as dressed spins with exponential tails); because extensively many of these appear, they provide a basis for approximate diagonalization of the system. In this way MBL systems avoid thermalization by a mechanism related to integrable systems, with conservation laws arising from (and particular to) a disorder realization.

# 4 Assignment: quantum statistical mechanics and thermalization

## 4.1 Dynamical ETH

Set up dense exact diagonalization for the quantum Ising model in a field with both transverse and longitudinal components:

$$H = -J \sum_{j=1}^{L} \sigma_j^z \sigma_{j+1}^z - h^x \sum_{j=1}^{L} \sigma_j^x - h^z \sum_{j=1}^{L} \sigma_j^z \tag{8}$$

I think the $\sigma_z$ should change with time because we have:

$$\frac{d}{dt}\sigma_z = i[H, \sigma_z] \tag{1}$$

but $\sigma_z$ does not commute with the Hamiltonian because it does not commute with the term that contains $\sigma_x$.

For simplicity, set $J = 1$. You should use periodic b.c. throughout the assignment to minimize finite-size effects. We are not so concerned with the ground state of the system or its low-energy states; what is important is that any $h^z \neq 0$ breaks the integrability of the transverse-field Ising model, thus we expect ETH behavior. We will use the particular values $(h^x, h^z) = (-1.05, 0.5)$, which specify a sufficiently generic point in the parameter space.

```python
def periodic_dense_hamiltonian(L, h_x, h_z, J=1):
    # Initialize Hamiltonian to zero matrix
    H = np.zeros((2 ** L, 2 ** L))

    # Define Pauli matrices
    sigma_x = np.array([[0, 1], [1, 0]])
    sigma_z = np.array([[1, 0], [0, -1]])
    I = np.identity(2)

    # add the tensor product helper function
    def tensor_product(matrices):
        """Calculate the tensor product of a list of matrices."""
        result = matrices[0]
        for matrix in matrices[1:]:
            result = np.kron(result, matrix)
        return result

    # Interaction term
```

```
    for i in range(L):  # Add periodic term at the end if periodic
        matrices = [I] * L  # Start with identity matrices
        matrices[i] = sigma_z  # Apply sigma_z at position i
        matrices[(i + 1) % L] = sigma_z  # Apply sigma_z at position (i+1) modulo L for
    periodic
        H += -J * tensor_product(matrices)

    # Transverse field term for x
    for i in range(L):
        matrices = [I] * L  # Start with identity matrices
        matrices[i] = sigma_x  # Apply sigma_x at position i
        H += -h_x * tensor_product(matrices)

    # Transverse field term for z
    for i in range(L):
        matrices = [I] * L
        matrices[i] = sigma_z
        H += -h_z * tensor_product(matrices)

    return H
```

### 4.1.1 Time evolution of an initial state

We wish to choose an initial state with average energy from somewhere near the middle of the energy spectrum. As thermalization is a general feature, this choice should not be too important, but it can affect the early-time behavior, or the time required to reach the thermal state. For simplicity, use a translation-invariant product state:

$$|\psi(t=0)\rangle = |\xi\rangle_1 \otimes |\xi\rangle_2 \otimes \cdots \otimes |\xi\rangle_L, \quad |\xi\rangle = \frac{1}{2}(|\uparrow\rangle - \sqrt{3}|\downarrow\rangle) \tag{9}$$

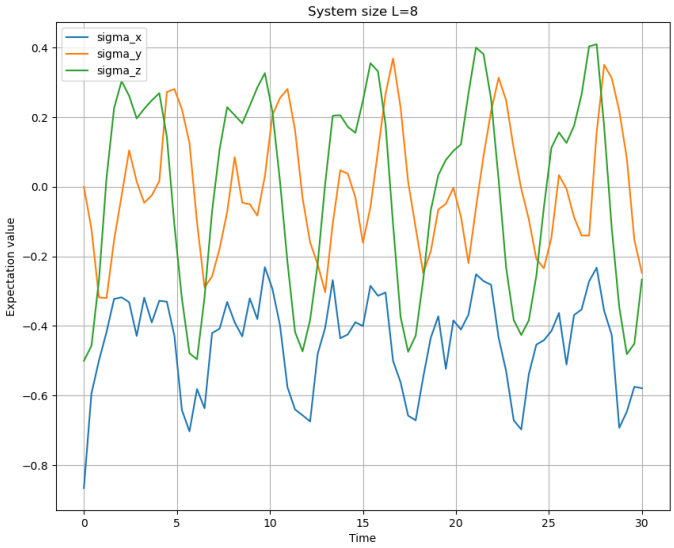
First, diagonalize $H$ for the parameter values specified in the previous section using various system sizes $L = 8, 10, 12, 14$. Use the eigenstates of $H$ you computed to expand the initial state $|\psi(t=0)\rangle$ in the energy eigenbasis. That is, calculate the coefficients $c_n$ of $|\psi(t=0)\rangle = \sum_{n=1}^{2^L} c_n|n\rangle$, where $n$ indexes the eigenstates $|n\rangle$ of $H$, with energy eigenvalues $\varepsilon_n$. At this point we could compute the time-evolved state according to (2); however, it turns out we do not even need to explicitly perform this calculation. We can instead directly evolve the expectation value itself in time via the expression (3). Use this method, and plot the time dependence of the observables $\langle\sigma_1^\mu(t)\rangle$, $\mu = x, y, z$, for each system size $L$ (the choice of site is arbitrary, due to translation invariance). Time-evolve for long enough to observe a qualitative change in the observables' behavior, using a short enough step size $\delta t$ to give a smooth curve. Based on your data, what behavior do you expect as $L \to \infty$ ? At late times you may observe a seeming return to a state like the initial one. This is not a Poincaré recurrence - which occurs only after doubly-exponentially long times - but rather a finite-size effect arising from "wavefronts" reaching the boundaries of the system and bouncing back and forth, or circling the entire system in the periodic case.

```python
def compute_observable_expectation(t, observable, overlap_coefficients, eigenvalues, eigenvectors):
    """
    Compute the time-dependent expectation value of an observable.

    Parameters:
    - t (float): Time at which to compute the expectation.
    - observable (np.ndarray): The observable matrix.
    - overlap_coefficients (np.ndarray): Coefficients of the initial state in the energy eigenbasis.
    - eigenvalues (np.ndarray): Array of eigenvalues from the Hamiltonian diagonalization.
    - eigenvectors (np.ndarray): Array of eigenvectors from the Hamiltonian diagonalization.

    Returns:
    - expectation (complex): The expectation value of the observable at time t.
    """
    # Calculate matrix elements in the eigenbasis
    matrix_element = eigenvectors.conj().T @ observable @ eigenvectors

    # Calculate phase differences using broadcasting
    phase = np.exp(-1j * (eigenvalues[:, None] - eigenvalues[None, :]) * t)

    # Reshape overlap coefficients for broadcasting
    overlap_coefficients = overlap_coefficients[:, None]

    # Calculate expectation value
    expectation = np.sum(overlap_coefficients.conj() * overlap_coefficients.T * phase * matrix_element)

    return expectation
```

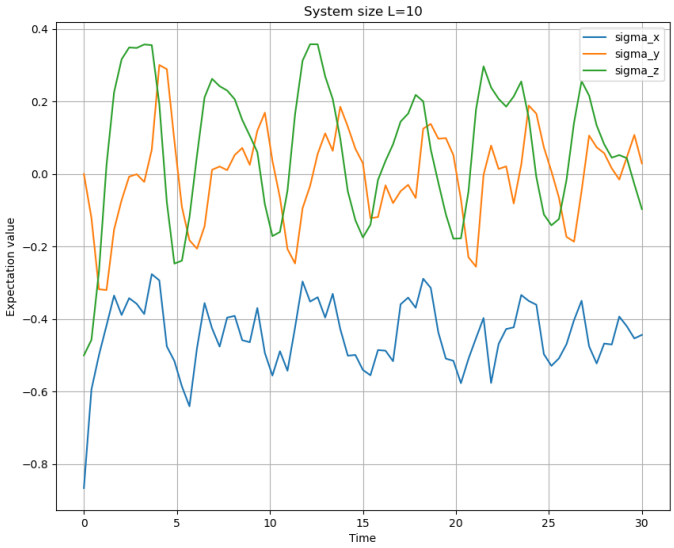
I was only able to do system sizes of 8 and 10. A dampening effect can be seen as we move towards longer times for the larger system sizes. The presence of this dampening effect was more prominent for larger system sizes, so I expect that the behavior in the infinite system limit would be to approach the thermal equilibrium value for the given observable at large times.

```python
# System size and parameters
import numpy as np
import matplotlib.pyplot as plt
from hw3.src.p4_1.fns import compute_observable_expectation, periodic_dense_hamiltonian, make_product_state
from hw1.src.hw1 import tensor_product

L_values = [8, 10]
h_x = -1.05
h_z = 0.5
t_values = np.linspace(0, 30, 75)

# Define the observables for the first site
sigma_x = np.array([[0, 1], [1, 0]])
sigma_y = np.array([[0, -1j], [1j, 0]])
sigma_z = np.array([[1, 0], [0, -1]])
identity = np.identity(2)

observables_labels = ['sigma_x', 'sigma_y', 'sigma_z']

```

```
def extend_observables(L):
    """Extend observables to the full system size."""
    full_observables = {
        'sigma_x': tensor_product([sigma_x] + [identity] * (L - 1)),
        'sigma_y': tensor_product([sigma_y] + [identity] * (L - 1)),
        'sigma_z': tensor_product([sigma_z] + [identity] * (L - 1))
    }
    return full_observables

# Ensure normalization

# Loop over different system sizes
for L in L_values:
    # Extend observables to the full system size
    full_observables = extend_observables(L)

    # Initial state: tensor product of single_site across all sites
    single_site = np.array([1, -np.sqrt(3)]) / 2
    initial_state = make_product_state(single_site, L)

    # Prepare to plot
    plt.figure(figsize=(10, 8))
    plt.title(f"System size L={L}")
    plt.xlabel("Time")
    plt.ylabel("Expectation value")

    # Generate the Hamiltonian
    H = periodic_dense_hamiltonian(L, h_x, h_z)

    # Diagonalize the Hamiltonian
    eigenvalues, eigenvectors = np.linalg.eigh(H)

    # Calculate the overlap coefficients
    overlap_coefficients = np.dot(eigenvectors.conj().T, initial_state)

    for label, observable in full_observables.items():
        expectations = []
        for t in t_values:
            expectation = compute_observable_expectation(t, observable,
    overlap_coefficients, eigenvalues, eigenvectors)
            expectations.append(np.real(expectation))  # Using real part; adjust if
    needed

        plt.plot(t_values, expectations, label=label)

    plt.legend()
    plt.grid(True)
    plt.savefig(f"hw3/docs/images/p4_1_1_expectations_L{L}.png")
```




(a) L=8




(b) L=10

Figure 1: Expectation values of the observables $\langle \sigma_1^{\mu}(t) \rangle$ for different values of L

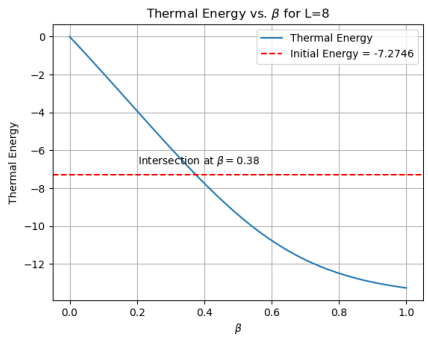
### 4.1.2 Thermal values of observables

It is evident from (2) that the energy of the state $E = \langle\psi(0)|H|\psi(0)\rangle$ does not change with time, and in fact energy is the only conserved quantity in the Hamiltonian (8). Thus we should use $E$ to determine the asymptotic temperature of the equilibrium thermal state. To do so, plot the thermal state energy

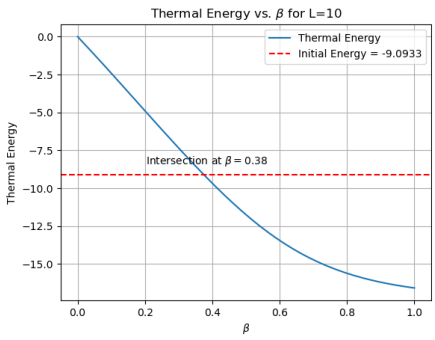
$$E_\beta = \frac{1}{Z_\beta}\,\mathrm{tr}\left[He^{-\beta H}\right] = \frac{1}{Z_\beta}\sum_n e^{-\beta\varepsilon_n}\langle n|H|n\rangle = \frac{1}{Z_\beta}\sum_n e^{-\beta\varepsilon_n}\varepsilon_n \tag{10}$$

As a function of $\beta$ for each $L$, you can then match these values with the energy $E$ of the initial state in order to determine the corresponding inverse temperature $\beta$ of the equilibrium state.

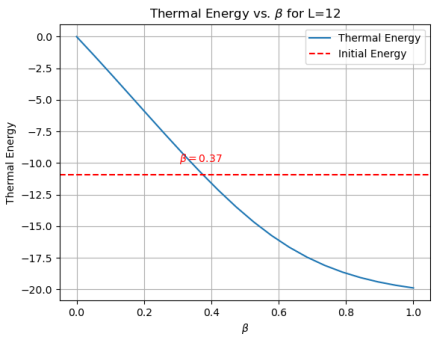
I computed the thermal energy as a function of beta and then I computed the initial state energy that is conserved in order to determine the corresponding inverse temperature beta of the equilibrium state.



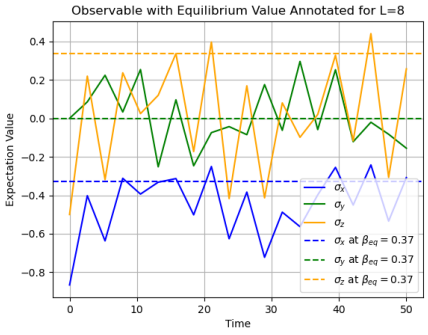

(a) Thermal energy L=8



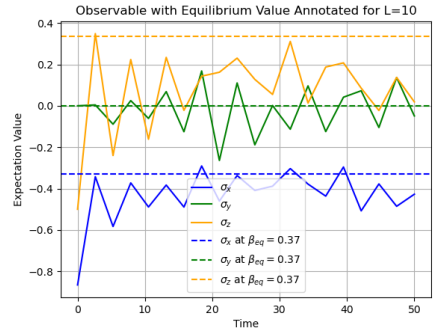

(b) Thermal energy L=10



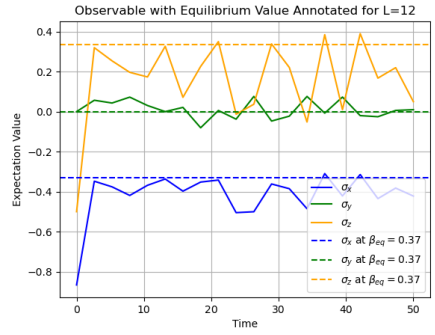

(c) Thermal energy L=12




(d) Observables annotation L=8




(e) Observables annotation L=10




(f) Observables annotation L=12

Figure 2: Thermal values of observables and their annotations.

```python
import numpy as np
import matplotlib.pyplot as plt
from hw3.src.p4_1.fns import compute_observable_expectation, periodic_dense_hamiltonian,
    make_product_state, time_dependent_state
from hw2.src.p5_2 import entanglement_entropy, calculate_reduced_density_matrix

# Set system parameters
L_values = [8, 10, 12]
h_x = -1.05
h_z = 0.5
t_values = np.linspace(0, 50, 20)

for L in L_values:
    # Generate the Hamiltonian
    H = periodic_dense_hamiltonian(L, h_x, h_z)
    # Diagonalize the Hamiltonian
    eigenvalues, eigenvectors = np.linalg.eigh(H)

    # Initial state: tensor product of single_site across all sites
    single_site1 = np.array([1, -np.sqrt(3)]) / 2
    initial_state1 = make_product_state(single_site1, L)
    # make a second state
    single_site2 = np.array([-2, 1]) / np.sqrt(5)
    initial_state2 = make_product_state(single_site2, L)

    # Prepare to plot
```

```
    plt.figure(figsize=(10, 8))
    plt.title(f"System size L={L}")
    plt.xlabel("Time")
    plt.ylabel("Entanglement entropy")

    # Calculate the overlap coefficients
    overlap_coefficients1 = np.dot(eigenvectors.conj().T, initial_state1)
    overlap_coefficients2 = np.dot(eigenvectors.conj().T, initial_state2)

    # initialize a list of entropy values
    entropy_values1 = []
    entropy_values2 = []

    for t in t_values:
        # Compute the time-dependent state
        state1 = time_dependent_state(t, overlap_coefficients1, eigenvalues, eigenvectors
)
        state2 = time_dependent_state(t, overlap_coefficients2, eigenvalues, eigenvectors
)

        # Compute the reduced density matrix
        reduced_density_matrix1 = calculate_reduced_density_matrix(state1, L, L // 2)
        reduced_density_matrix2 = calculate_reduced_density_matrix(state2, L,  L // 2)

        # Compute the entanglement entropy
        entropy1 = entanglement_entropy(reduced_density_matrix1)
        entropy2 = entanglement_entropy(reduced_density_matrix2)

        entropy_values1.append(entropy1)
        entropy_values2.append(entropy2)

    plt.plot(t_values, entropy_values1, label="Entanglement entropy for state 1")
    plt.plot(t_values, entropy_values2, label="Entanglement entropy for state 2")
    plt.legend()
    plt.grid()
    plt.savefig(f"hw3/docs/images/p4_1_3_Entanglement_Entropy_L={L}.png")
```

Now measure the same observables $\langle \sigma_1^\mu \rangle_\beta, \mu = x, y, z$, in the equilibrium state using (6) and compare these to your time-dependent values by marking these values as horizontal lines on your time trace of $\langle \sigma_1^\mu(t) \rangle$. Comment on the approach to equilibrium and the dependence on system size. You will notice that one of the $\langle \sigma_1^\mu \rangle_\beta$ disappears identically. Which one is this, and why?

Again, as time goes on especially with larger system sizes, there is an approach to the equilibrium value for the observables, which is the same phenomenon as the damping that I mentioned earlier. However, this was less pronounced for the smaller system sizes that I was able to reach. I notice that the $\langle \sigma_1^y \rangle_\beta$ observable disappears identically. This can be justified as follows:

$$\langle \Psi | \sigma_1^y | \Psi \rangle = i \langle \Psi | \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} | \Psi \rangle = i\mathbb{R} \tag{2}$$

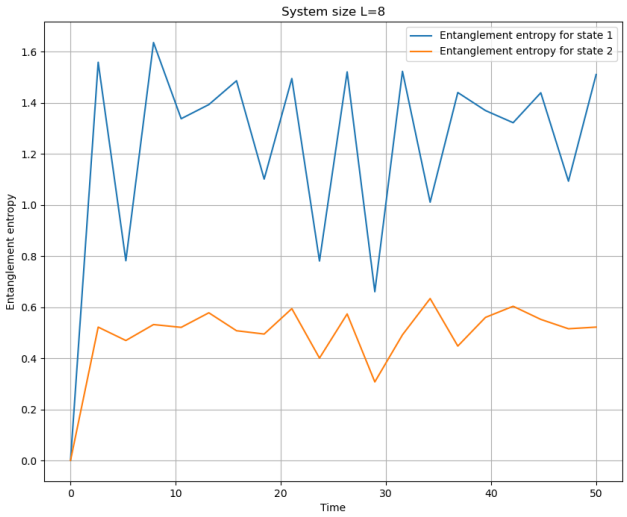
so this observable has a purely imaginary expectation value, and so it shows up as a horizontal line with a value of 0 in the plot.

### 4.1.3 Entanglement entropy growth with time

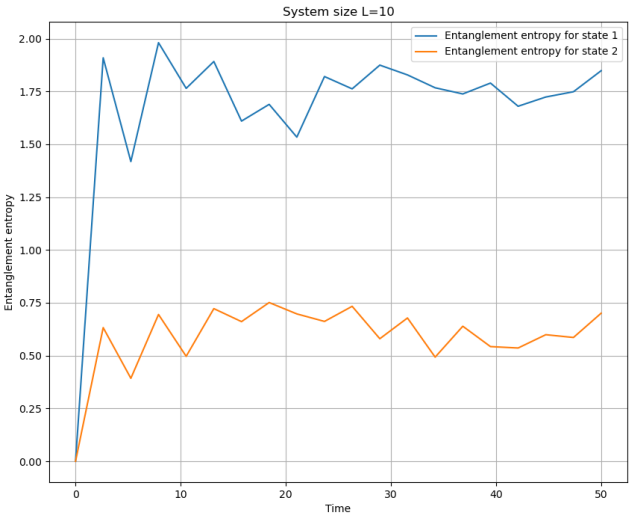
The initial state $|\psi(0)\rangle$ has highly excited energy, lying near the middle of the spectrum. However, because it's a product state and therefore very atypical, one may wonder whether under unitary time evolution the property of low entanglement might be preserved in any way. Compute the time-dependent state $|\psi(t)\rangle$ using (2) and measure the half-system entanglement entropy $S_{L/2}(t)$. Plot the time trace of this quantity, and comment on the behavior at both early and late times. To ensure that we've not accidentally chosen a special excited state, repeat this experiment for some other product state. What are the implications of your measurements with regard to our ability to simulate the time evolution of quantum states using MPS?

Since a product state is qualitatively similar to the MPS list in its ability to be decomposed into a tensor on any given site, and the product state seems to be able to represent time evolution of the entanglement entropy well with large jumps at initial times and converging towards an equilibrium value at later times, I would expect the MPS list to be able to do the same.

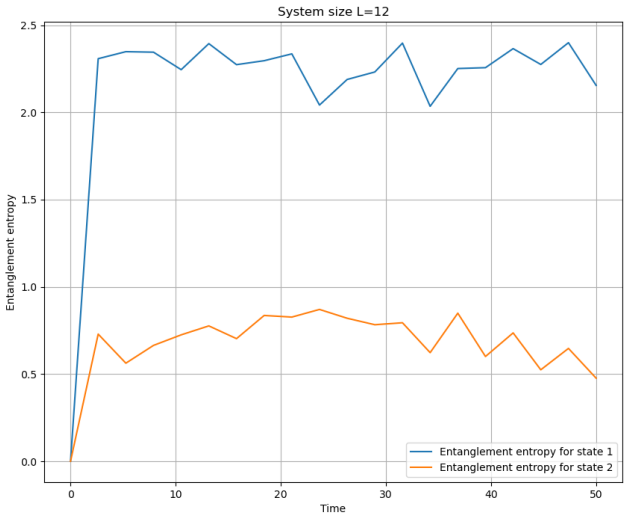
```python
def time_dependent_state(t, overlap_coefficients, eigenvalues, eigenvectors):
    """Computes the time-dependent state |\psi(t)>."""
    return np.sum(np.exp(-1j * eigenvalues * t) * overlap_coefficients * eigenvectors, axis=1)

import numpy as np
import matplotlib.pyplot as plt
from hw3.src.p4_1.fns import compute_observable_expectation, periodic_dense_hamiltonian, make_product_state, time_dependent_state
from hw2.src.p5_2 import entanglement_entropy, calculate_reduced_density_matrix

# Set system parameters
L_values = [8, 10, 12]
h_x = -1.05
h_z = 0.5
t_values = np.linspace(0, 50, 20)

for L in L_values:
    # Generate the Hamiltonian
    H = periodic_dense_hamiltonian(L, h_x, h_z)
    # Diagonalize the Hamiltonian
    eigenvalues, eigenvectors = np.linalg.eigh(H)

    # Initial state: tensor product of single_site across all sites
    single_site1 = np.array([1, -np.sqrt(3)]) / 2
    initial_state1 = make_product_state(single_site1, L)
    # make a second state
    single_site2 = np.array([-2, 1]) / np.sqrt(5)
    initial_state2 = make_product_state(single_site2, L)

    # Prepare to plot
    plt.figure(figsize=(10, 8))
    plt.title(f"System size L={L}")
    plt.xlabel("Time")
    plt.ylabel("Entanglement entropy")

    # Calculate the overlap coefficients
    overlap_coefficients1 = np.dot(eigenvectors.conj().T, initial_state1)
    overlap_coefficients2 = np.dot(eigenvectors.conj().T, initial_state2)

    # initialize a list of entropy values
    entropy_values1 = []
    entropy_values2 = []

    for t in t_values:
        # Compute the time-dependent state
        state1 = time_dependent_state(t, overlap_coefficients1, eigenvalues, eigenvectors)
        state2 = time_dependent_state(t, overlap_coefficients2, eigenvalues, eigenvectors)

        # Compute the reduced density matrix
        reduced_density_matrix1 = calculate_reduced_density_matrix(state1, L, L // 2)
```




(a) L=8




(b) L=10




(c) L=12

Figure 3: Entanglement Entropy for different values of L

```
        reduced_density_matrix2 = calculate_reduced_density_matrix(state2, L,  L // 2)

        # Compute the entanglement entropy
        entropy1 = entanglement_entropy(reduced_density_matrix1)
        entropy2 = entanglement_entropy(reduced_density_matrix2)

        entropy_values1.append(entropy1)
        entropy_values2.append(entropy2)

    plt.plot(t_values, entropy_values1, label="Entanglement entropy for state 1")
    plt.plot(t_values, entropy_values2, label="Entanglement entropy for state 2")
    plt.legend()
    plt.grid()
    plt.savefig(f"hw3/docs/images/p4_1_3_Entanglement_Entropy_L={L}.png")
```

## 4.2 Eigenstate ETH

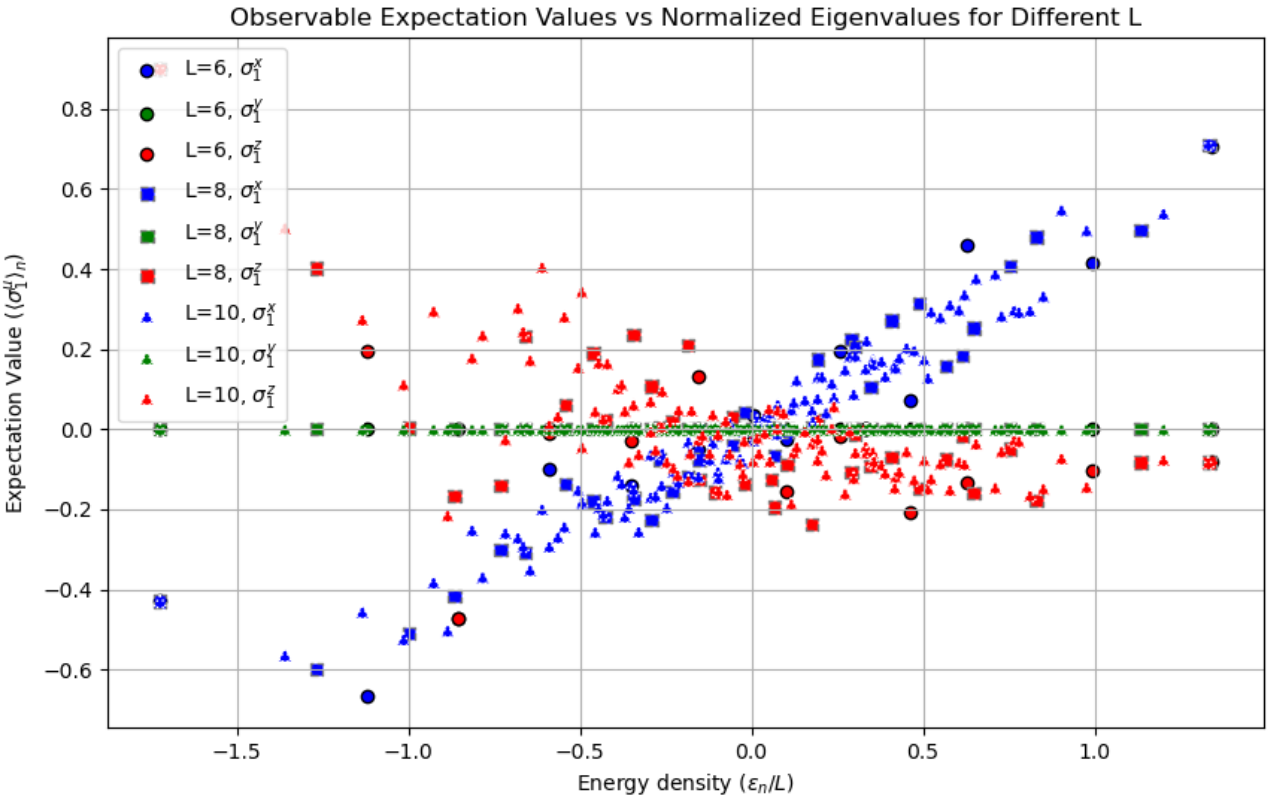
### 4.2.1 Observables in excited states

In Sec. 4.1 .1 you computed the expectation values $\langle \sigma_1^\mu \rangle_n, \mu = x, y, z$, in each eigenstate $n$ of the Hamiltonian. ETH predicts that the value of each observable depends only on the energy of the eigenstate, indirectly through the temperature of the Gibbs ensemble. To confirm this we can plot the expectation values of the observables directly as a function of the energy eigenvalue $\varepsilon_n$.
The situation is different now, because we want to consider the expectation values in the eigenstates of the Hamiltonian, not the time-evolved state. This looks like:

$$\langle \sigma_1^\mu \rangle_n = \langle n | \, \sigma_1^\mu \, | n \rangle \tag{3}$$

First however, recall that the initial state $|\psi(0)\rangle$ is translation invariant. Under a translation invariant Hamiltonian on a periodic system, momentum is well-defined; this state occupies the
$k = 0$ momentum sector. You may have noticed that many of the $c_n$ in the expansion of $|\psi(0)\rangle$ in the eigenbasis were exactly 0 ; these correspond to eigenstates in other momentum sectors, which are orthogonal to this state. Hence we want to consider only eigenstates with $k = 0$; to identify these we could in principle observe the overlaps $c_n$, but this depends on specific details of the initial state and is difficult for larger $L$. A better method is use the translation operator $T$, which shifts the entire system by 1 site: $T \, |\sigma_1, \sigma_2, \sigma_3 \ldots, \sigma_L\rangle = |\sigma_L, \sigma_1, \sigma_2, \ldots, \sigma_{L-1}\rangle$. The $k = 0$ sector can be identified by $\langle n|T|n\rangle = +1$; a matrix element that is any other phase, including -1 , indicates a state from another sector.

Using $T$ to filter the eigenstates, plot the expectation values $\langle \sigma_1^\mu \rangle_n$ for all $|n\rangle$ in the $k = 0$ momentum sector as a function of $\varepsilon_n/L$. Plot the data on top of each other for the various system sizes $L$. How does the behavior depend on $\varepsilon_n/L$, and how does this change with increasing $L$ ?




As we increase the energy density, we can observe 3 trends. As noted earlier, the $\sigma_y$ Pauli matrix is purely imaginary, so its expectation value is constant at 0. Next, we can consider $\sigma_x$; since we set $h^x$ to -1.05. At the band edge with the lowest energy density (this can be thought of as the "ground state of the phase"), we observe that the value of this expectation is the most negative, while at the right-hand band edge with the highest energy density, the expectation value is the most positive since now we are going against the negative value of the field strength in the x direction. The situation is opposite for the $\sigma_z$ observable, where the expectation value is the most positive at the lowest energy density and the most negative at the highest energy density; by a similar reasoning to the one that was given before, this is because the $h^z$ was set to 0.5. These trends seem to be consistent across system sizes, but we note that for the larger system sizes there are more eigenvalues in the $k = 0$ sector and so we see more data points.

```python
def translation_operator(L):
    """Construct the translation operator T for a system of size L."""
    T = np.zeros((2 ** L, 2 ** L))

    for i in range(2 ** L):
        state = format(i, f'0{L}b')  # Binary representation of the state
        new_state = state[-1] + state[:-1]  # Shift by one site to the right
        new_index = int(new_state, 2)  # Convert back to decimal
        T[i, new_index] = 1

    return T

def identify_k0_sector(eigenvectors, T):
    """
    Identify eigenstates in the k=0 sector using the translation operator T.

```

```
    Parameters:
    - eigenvectors: Array of eigenvectors of the Hamiltonian.
    - T: Translation operator.

    Returns:
    - k0_indices: Indices of the k=0 sector eigenstates.
    """
    k0_indices = []
    for n, eigenvector in enumerate(eigenvectors.T):
        overlap = np.dot(eigenvector.conj().T, np.dot(T, eigenvector))
        if np.isclose(overlap, 1.0, atol=1e-8):
            k0_indices.append(n)
    return k0_indices
def compute_observable_expectation_eigenvalue(eigenindex, observable, eigenvectors):
    """
    Compute the expectation value of an observable given the eigenindex.

    Parameters:
    - eigenindex: The index of the eigenvalue.
    - observable: The observable matrix.
    - eigenvalues: Eigenvalues from the Hamiltonian diagonalization.
    - eigenvectors: Eigenvectors from the Hamiltonian diagonalization.

    Returns:
    - expectation: The expectation value of the observable.
    """
    expectation = np.dot(eigenvectors[:, eigenindex].T, np.dot(observable, eigenvectors
    [:, eigenindex]))
    return expectation
import matplotlib.pyplot as plt
import numpy as np
from hw3.src.p4_2.fns import translation_operator, identify_k0_sector,
    compute_observable_expectation_eigenvalue
from hw3.src.p4_1.fns import periodic_dense_hamiltonian
from hw1.src.hw1 import tensor_product

# Define scatter styles, line styles, and colors for different system sizes and
    observables
scatter_styles = ['o', 's', '^']
edge_colors = ['black', 'gray', 'white']  # Edge colors for different system sizes
line_styles = ['-', '--', ':']
colors = ['blue', 'green', 'red']  # Colors for sigma_x, sigma_y, sigma_z
observable_labels = ['x', 'y', 'z']

# System sizes
L_values = [6, 8, 10]

# Constants
h_x = -1.05
h_z = 0.5

# Prepare the plot
plt.figure(figsize=(10, 6))
plt.title('Observable Expectation Values vs Normalized Eigenvalues for Different L')
plt.xlabel(r'Energy density ($\epsilon_n / L$)')
plt.ylabel(r'Expectation Value ($\langle\sigma_1^\mu\rangle_n$)')

# Loop over system sizes
```

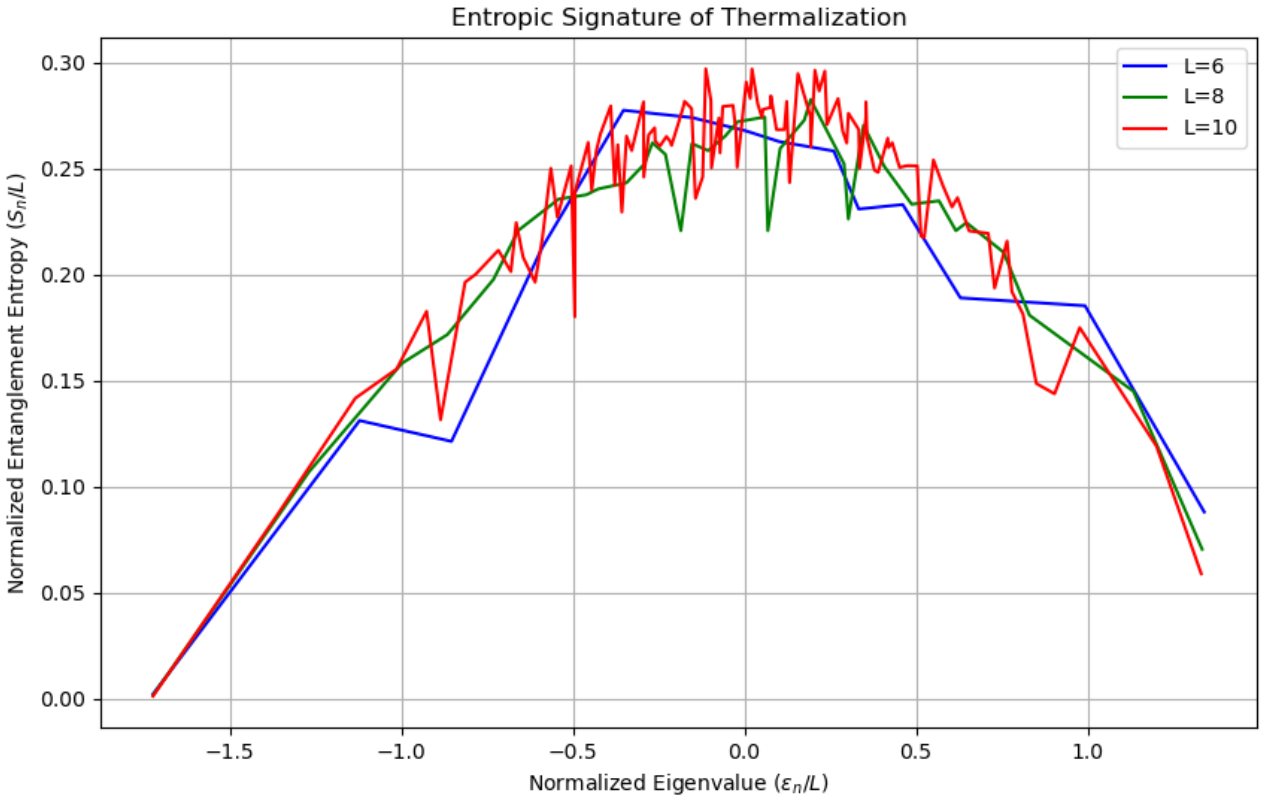
```
for i, L in enumerate(L_values):
    # Generate and diagonalize the Hamiltonian
    H = periodic_dense_hamiltonian(L, h_x, h_z)
    eigenvalues, eigenvectors = np.linalg.eigh(H)

    # Define the translation operator for the system
    translation_op = translation_operator(L)

    # Identify the k=0 sector
    k0_sector = identify_k0_sector(eigenvectors, translation_op)

    # Define observables
    sigma_x = np.array([[0, 1], [1, 0]])
    sigma_y = np.array([[0, -1j], [1j, 0]])
    sigma_z = np.array([[1, 0], [0, -1]])
    identity = np.identity(2)
    full_observables = [tensor_product([sigma] + [identity] * (L - 1)) for sigma in [
   sigma_x, sigma_y, sigma_z]]

    # Compute and plot expectation values for each observable
    for j, observable in enumerate(full_observables):
        expectation_values = []
        eigenvalues_k0 = []
        for _, k0_index in enumerate(k0_sector):
            observable_k0 = compute_observable_expectation_eigenvalue(k0_index,
   observable, eigenvectors)
            expectation_values.append(observable_k0)
            eigenvalues_k0.append(eigenvalues[k0_index])

        plt.scatter(
            np.array(eigenvalues_k0) / L,
            expectation_values,
            label=f'L={L}, $\sigma_{{1}}^{{ {observable_labels[j]} }}$',
            color=colors[j],
            marker=scatter_styles[i],
            edgecolors=edge_colors[i],
            linestyle=line_styles[i]
        )

# Add legend and show plot
plt.legend()
plt.grid(True)
plt.savefig("hw3/docs/images/p4_2_1_filtered_expectations.png")
plt.show()
```

[ADDED: A couple of technical remarks on using T: First, you might wonder why you are finding normalized Hamiltonian eigenstates $|n\rangle$ with $|\langle n|T|n\rangle| < 1$, which means that they are not eigenstates of the translation operator and seems to be contradiction. These can arise here because states with $k$ and $-k$ are degenerate, and without resolving the momentum explicitly, your black-box diagonalizer will likely find superpositions of these; observing $|\langle n|T|n\rangle| < 1$ is thus also a signature that these are not states with zero momentum. Second, to be precise in the ETH analysis, one needs to resolve also the inversion symmetry of the Hamiltonian, e.g., inversion in the middle point of the system, $I\,|\sigma_1, \sigma_2, \ldots, \sigma_{L-1}, \sigma_L\rangle = |\sigma_L, \sigma_{L-1}, \ldots, \sigma_2, \sigma_1\rangle$ (all other inversions can be generated by combining this with translations) and restrict to specific inversion quantum number sector, here $I = +1$ since this is the quantum number of $|\psi(0)\rangle$. Since this is expected to have a smaller effect than resolving the momentum, it is ok not to worry about the inversion symmetry in this assignment, or you can do a more brute-force treatment by filtering the energy eigenstates by their non-zero vs zero overlap with $|\psi(0)\rangle$ (remembering that the computer is unlikely to return exact zero, but a number close to machine precision is likely an exact zero).]

Optional. Also compare these results (for $L = 14$ ) with $\langle \sigma_1^\mu \rangle\,(\beta)$, the expectation values in the thermal state at inverse temperature $\beta$, plotted versus $E(\beta)/L = \langle H \rangle(\beta)/L$ as you vary $\beta$. This time when you calculate finite-temperature ensemble averages, use only $k = 0$ states for both the numerator and denominator $Z(\beta)$ in (6). This may help to reduce finite-size effects in the calculation. Which values of $\beta$ are associated with which regions of the energy spectrum?

### 4.2.2 Entropic signature of thermalization

In addition to the measurements of local observables in eigenstates, compute the half-system entanglement entropy for all $k = 0$ sector eigenstates and plot the quantity $S_{L/2}/L$ as a function of $\varepsilon_n/L$ for all system sizes $L$. What dependence on the energy density do you observe, and how does this depend on system size?




The entanglement entropy shows a symmetric trend; it has its lowest values at the band edges and its highest values in the middle of the energy spectrum. This statement is not entirely analogous to this situation, but if we can interpret the extrema of the energy density as the ground states of the ferromagnetic and paramagnetic phases, then the entanglement entropy is at its lowest in the ground states and at its highest in the middle of the energy spectrum. I see a similar trend across system sizes, but the larger system sizes have more eigenstates in the $k = 0$ sector, so we see more data points.

```
import matplotlib.pyplot as plt
import numpy as np
from hw3.src.p4_2.fns import translation_operator, identify_k0_sector
from hw3.src.p4_1.fns import periodic_dense_hamiltonian
from hw1.src.hw1 import tensor_product
from hw2.src.p5_2 import entanglement_entropy, calculate_reduced_density_matrix
# Define line styles and colors for different system sizes and observables
line_styles = ['-', '--', ':']
colors = ['blue', 'green', 'red']  # Colors for sigma_x, sigma_y, sigma_z
observable_labels = ['x', 'y', 'z']

# System sizes
L_values = [6, 8, 10]

# Constants
h_x = -1.05
h_z = 0.5

# Prepare the plot
plt.figure(figsize=(10, 6))
plt.title('Entropic Signature of Thermalization')
```

```
plt.xlabel(f'Normalized Eigenvalue ($\epsilon_n / L$)')
plt.ylabel(f'Normalized Entanglement Entropy ($S_n / L$)')

# Loop over system sizes
for i, L in enumerate(L_values):
    # Generate and diagonalize the Hamiltonian
    H = periodic_dense_hamiltonian(L, h_x, h_z)
    eigenvalues, eigenvectors = np.linalg.eigh(H)

    # Define the translation operator for the system
    translation_op = translation_operator(L)

    # Identify the k=0 sector
    k0_sector = identify_k0_sector(eigenvectors, translation_op)

    expectation_values = []
    eigenvalues_k0 = []
    for _, k0_index in enumerate(k0_sector):
        # Compute the reduced density matrix
        reduced_density_matrix = calculate_reduced_density_matrix(eigenvectors[:,
   k0_index], L, L // 2)
        # Compute the entanglement entropy
        entropy = entanglement_entropy(reduced_density_matrix)
        expectation_values.append(entropy)
        eigenvalue_k0 = eigenvalues[k0_index]
        eigenvalues_k0.append(eigenvalue_k0)

    plt.plot(np.array(eigenvalues_k0) / L, np.array(expectation_values) / L, color=colors
   [i])
    # add a label for the color of the curve
    plt.plot([], [], color=colors[i], label=f'L={L}')

# Add legend and show plot
plt.legend()
plt.grid()
plt.savefig(f"hw3/docs/images/p4_2_2_entropic_signature.png")
```

Optional. We have been borrowing intuition throughout this assignment from equilibrium statistical mechanics to describe quantum systems. We can test the wisdom of this approach by comparing quantum entanglement entropy with entropy in its original sense: thermodynamic entropy,

which in the canonical ensemble takes the form $S_{\text{th}}(\beta) = -\sum_n w_n^{(\beta)} \log w_n^{(\beta)}$, $w_n^{(\beta)} = e^{-\beta\varepsilon_n}/Z_\beta$. For $L = 14$, plot both $S_{L/2}/(L/2)$ versus $\varepsilon_n/L$ for eigenstates $n$ as well as the thermodynamic entropy $S_{\text{th}}(\beta)/L$ versus $\langle H\rangle(\beta)/L$ by varying $\beta$. Is it appropriate to use the same word "entropy" to describe both quantities?

## 4.3 Violations of ETH

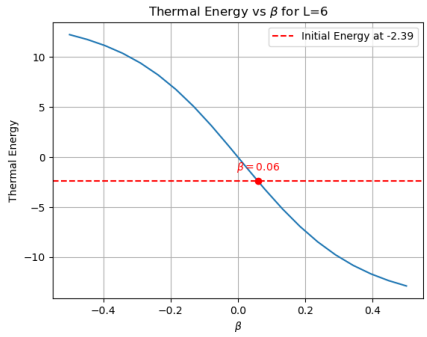
### 4.3.1 Many-body localized model

We now introduce quenched disorder to the magnetic field terms in the Hamiltonian (8), resulting in an MBL phase. That is, consider

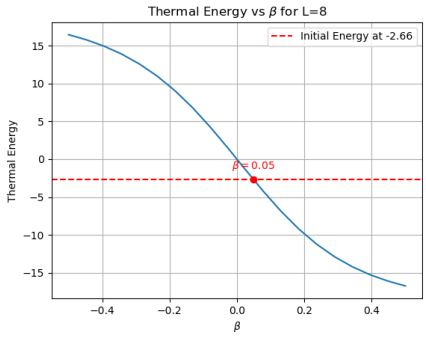
$$H = -J\sum_{j=1}^{L}\sigma_j^z\sigma_{j+1}^z - \sum_{j=1}^{L}h_j^x\sigma_j^x - \sum_{j=1}^{L}h_j^z\sigma_j^z \tag{11}$$

Again set $J = 1$, but now sample $h_j^x$ and $h_j^z$ independently from the uniform distribution over $[-W, W]$. For strong enough $W$ the model is MBL; for a start, you can try $W = 3$. Repeat the dynamics experiments you performed above for the ETH system, including the time evolution of observables with reference to the proper thermal state value. Note that the temperature of the "equilibrium" thermal state will be different from the previous case, and depends on your disorder realization.

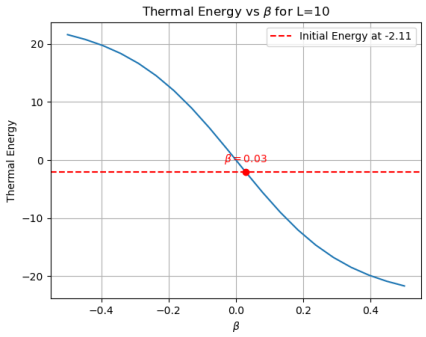
You should again measure local observables in the eigenstates as well as entanglement entropy, as you did for ETH (however, now there is no translation invariance, so no momentum sectors). You may wish to perform some averaging over disorder realizations for these eigenstate properties, in order to obtain a clearer picture of the behavior. Comment on the measurable difference between ETH and MBL physics.



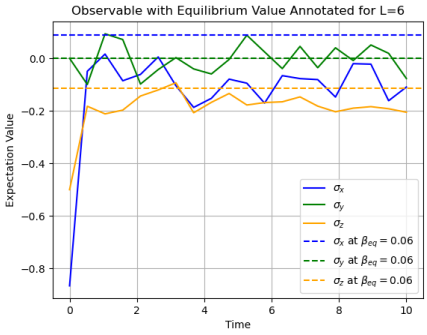

(a) Thermal energy L=6



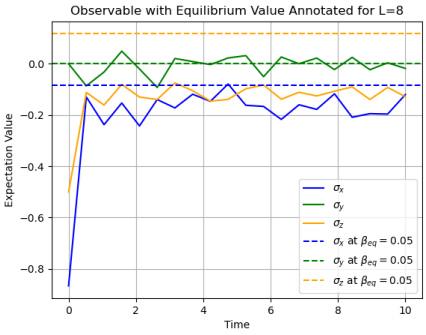

(b) Thermal energy L=8



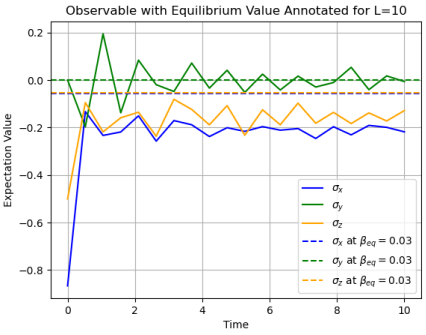

(c) Thermal energy L=10



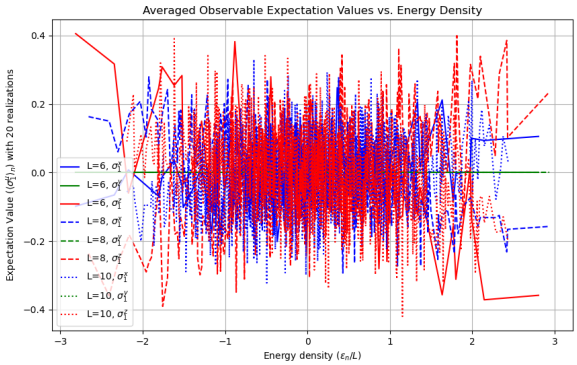

(d) Expectations L=6



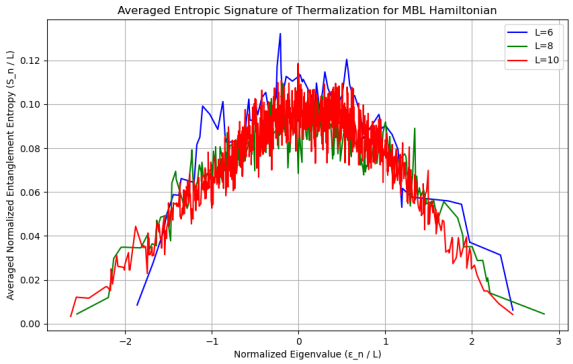

(e) Expectations L=8




(f) Expectations L=10




(g) Filtered Expectations Average




(h) Entropic Signature Average

Figure 4: Many-body localized model results.

I do observe the similar thermalization with time evolution, but only after I perform an average over multiple disorder realizations. However, we do see a chaotic observable expectation values as a function of energy density, even after performing multiple realizations for this calculation. This was not seen for the ETH model. The entanglement entropy also shows a similar trend to the ETH model, with lower values at the band edges.

```
def periodic_dense_hamiltonian_mbl(L, W, J=1):
    # Initialize Hamiltonian to zero matrix
    H = np.zeros((2 ** L, 2 ** L))

    # Define Pauli matrices
    sigma_x = np.array([[0, 1], [1, 0]])
    sigma_z = np.array([[1, 0], [0, -1]])
    I = np.identity(2)

```

```
    # add the tensor product helper function
    def tensor_product(matrices):
        """Calculate the tensor product of a list of matrices."""
        result = matrices[0]
        for matrix in matrices[1:]:
            result = np.kron(result, matrix)
        return result

    # Interaction term
    for i in range(L):  # Add periodic term at the end if periodic
        matrices = [I] * L  # Start with identity matrices
        matrices[i] = sigma_z  # Apply sigma_z at position i
        matrices[(i + 1) % L] = sigma_z  # Apply sigma_z at position (i+1) modulo L for periodic
        H += -J * tensor_product(matrices)

    # transverse field term for x
    for i in range(L):
        # sample from the uniform distribution defined by (-W, W) to get an h_x value
        h_x = np.random.uniform(-W, W)
        matrices = [I] * L
        matrices[i] = sigma_x
        H += -h_x * tensor_product(matrices)

    # transverse field term for z
    for i in range(L):
        # sample from the uniform distribution defined by (-W, W) to get an h_z value
        h_z = np.random.uniform(-W, W)
        matrices = [I] * L
        matrices[i] = sigma_z
        H += -h_z * tensor_product(matrices)

    return H
import numpy as np
import matplotlib.pyplot as plt
from hw1.src.hw1 import tensor_product
from hw3.src.p4_1.fns import compute_observable_expectation, make_product_state, compute_thermal_energy, compute_thermal_observable
from hw3.src.p4_3.fns import periodic_dense_hamiltonian_mbl

# Set system parameters
L_values = [4, 6, 8]
W = 3
beta_values = np.linspace(-0.5, 0.5, 20)
number_realizations = 20

# Define simple matrices
sigma_x = np.array([[0, 1], [1, 0]])
sigma_y = np.array([[0, -1j], [1j, 0]])
sigma_z = np.array([[1, 0], [0, -1]])
identity = np.identity(2)
observables_labels = [rf"$\sigma_{label}$" for label in ['x', 'y', 'z']]
observables_colors = ['blue', 'green', 'orange']

# Define time values once
t_values = np.linspace(0, 10, 20)

# Loop over different system sizes
```

```
for L in L_values:
    # Initialize arrays to store thermal energies and overlaps
    thermal_energies = np.zeros((number_realizations, len(beta_values)))
    initial_energies = np.zeros(number_realizations)
    observables_values_time = {label: np.zeros((number_realizations, len(t_values))) for label in observables_labels}

    # Generate the initial state
    single_site = np.array([1, -np.sqrt(3)]) / 2
    initial_state = make_product_state(single_site, L)

    # Extend observables to the full system size once
    full_sigma_x = tensor_product([sigma_x] + [identity] * (L - 1))
    full_sigma_y = tensor_product([sigma_y] + [identity] * (L - 1))
    full_sigma_z = tensor_product([sigma_z] + [identity] * (L - 1))
    full_observables = [full_sigma_x, full_sigma_y, full_sigma_z]

    for realization in range(number_realizations):
        # Generate the Hamiltonian
        H = periodic_dense_hamiltonian_mbl(L, W)
        # Diagonalize the Hamiltonian
        eigenvalues, eigenvectors = np.linalg.eigh(H)

        # Compute thermal energies for each beta using vectorized operations
        thermal_energies[realization, :] = np.array([compute_thermal_energy(beta, eigenvalues) for beta in beta_values])

        # Compute the initial energy of the initial state
        matrix_element = initial_state.conj().T @ H @ initial_state
        normalization = initial_state.conj().T @ initial_state
        initial_energy = matrix_element / normalization
        initial_energies[realization] = initial_energy.real

        # Calculate the overlap coefficients
        overlap_coefficients = np.dot(eigenvectors.conj().T, initial_state)

        # Compute time-dependent values for each observable
        for label, observable in zip(observables_labels, full_observables):
            expectations = np.array([
                compute_observable_expectation(t, observable, overlap_coefficients, eigenvalues, eigenvectors).real
                for t in t_values
            ])
            observables_values_time[label][realization, :] = expectations

    # Average thermal energies and initial energies over realizations
    avg_thermal_energies = np.mean(thermal_energies, axis=0)
    avg_initial_energy = np.mean(initial_energies)

    # Average time-dependent observable values over realizations
    avg_observables_values_time = {label: np.mean(observables_values_time[label], axis=0)
    for label in observables_labels}

    # Plot the horizontal dashed red line at the initial energy
    plt.figure()
    plt.axhline(y=avg_initial_energy, color='red', linestyle='--', label=rf"Initial Energy at {avg_initial_energy:.2f}")

```

```
    # Plot the results
    plt.title(f"Thermal Energy vs $\\beta$ for L={L}")
    plt.xlabel(r"$\beta$")
    plt.ylabel("Thermal Energy")
    plt.plot(beta_values, avg_thermal_energies)

    # Prepare data for interpolation
    sorted_indices = np.argsort(avg_thermal_energies)
    sorted_energies = avg_thermal_energies[sorted_indices]
    sorted_betas = beta_values[sorted_indices]

    # Interpolating to find intersection point
    beta_intersection = np.interp(avg_initial_energy, sorted_energies, sorted_betas)

    # Plot and annotate intersection point
    plt.plot(beta_intersection, avg_initial_energy, 'ro')
    plt.annotate(rf"$\beta = {beta_intersection:.2f}$", (beta_intersection,
    avg_initial_energy), textcoords="offset points", xytext=(0, 10), ha='center', color='
    red')
    plt.legend()
    plt.grid()
    plt.savefig(f"hw3/docs/images/p4_3_1_thermal_energy_intersection_L{L}.png")

    # Plotting section for observables
    plt.figure()
    plt.title(f"Observable with Equilibrium Value Annotated for L={L}")
    plt.xlabel("Time")
    plt.ylabel("Expectation Value")

    # Plot each observable as a line plot
    for label, color in zip(observables_labels, observables_colors):
        plt.plot(t_values, avg_observables_values_time[label], label=label, color=color)

    # Add a horizontal line for each observable
    for label, color in zip(observables_labels, observables_colors):
        thermal_observable = compute_thermal_observable(beta_intersection, eigenvalues,
    eigenvectors, full_observables[observables_labels.index(label)])
        plt.axhline(y=thermal_observable, color=color, linestyle='--', label=rf"{label}
    at $\beta_{{eq}} = {beta_intersection:.2f}$")

    plt.legend()
    plt.grid()
    plt.savefig(f"hw3/docs/images/p4_3_1_expectation_L{L}.png")

import numpy as np
import matplotlib.pyplot as plt
from hw3.src.p4_1.fns import make_product_state, time_dependent_state
from hw2.src.p5_2 import entanglement_entropy, calculate_reduced_density_matrix
from hw3.src.p4_3.fns import periodic_dense_hamiltonian_mbl

# Set system parameters
L_values = [6, 8, 10]
W = 3
t_values = np.linspace(0, 50, 20)
number_realizations = 10

for L in L_values:
    average_entropy_values1 = np.zeros(len(t_values))
```

```
    average_entropy_values2 = np.zeros(len(t_values))
    for realization in range(number_realizations):
        # Generate the Hamiltonian
        H = periodic_dense_hamiltonian_mbl(L, W)
        # Diagonalize the Hamiltonian
        eigenvalues, eigenvectors = np.linalg.eigh(H)

        # Initial state: tensor product of single_site across all sites
        single_site1 = np.array([1, -np.sqrt(3)]) / 2
        initial_state1 = make_product_state(single_site1, L)
        # make a second state
        single_site2 = np.array([-2, 1]) / np.sqrt(5)
        initial_state2 = make_product_state(single_site2, L)

        # Calculate the overlap coefficients
        overlap_coefficients1 = np.dot(eigenvectors.conj().T, initial_state1)
        overlap_coefficients2 = np.dot(eigenvectors.conj().T, initial_state2)

        # initialize a list of entropy values
        entropy_values1 = []
        entropy_values2 = []

        for t in t_values:
            # Compute the time-dependent state
            state1 = time_dependent_state(t, overlap_coefficients1, eigenvalues,
    eigenvectors)
            state2 = time_dependent_state(t, overlap_coefficients2, eigenvalues,
    eigenvectors)

            # Compute the reduced density matrix
            reduced_density_matrix1 = calculate_reduced_density_matrix(state1, L, L // 2)
            reduced_density_matrix2 = calculate_reduced_density_matrix(state2, L, L // 2)

            # Compute the entanglement entropy
            entropy1 = entanglement_entropy(reduced_density_matrix1)
            entropy2 = entanglement_entropy(reduced_density_matrix2)

            entropy_values1.append(entropy1)
            entropy_values2.append(entropy2)

        average_entropy_values1 += np.array(entropy_values1)
        average_entropy_values2 += np.array(entropy_values2)

    average_entropy_values1 /= number_realizations
    average_entropy_values2 /= number_realizations

    # Prepare to plot
    plt.figure(figsize=(10, 8))
    plt.title(f"System size L={L}")
    plt.xlabel("Time")
    plt.ylabel("Avg. Entanglement entropy")

    plt.plot(t_values, average_entropy_values1, label="Entanglement entropy for state 1")
    plt.plot(t_values, average_entropy_values2, label="Entanglement entropy for state 2")
    plt.legend()
    plt.grid()
    plt.savefig(f"hw3/docs/images/p4_3_1_avg_tdee_L={L}.png")
    plt.close()
```

```
229
230 import matplotlib.pyplot as plt
231 import numpy as np
232 from hw3.src.p4_2.fns import compute_observable_expectation_eigenvalue
233 from hw1.src.hw1 import tensor_product
234 from hw3.src.p4_3.fns import periodic_dense_hamiltonian_mbl
235
236 # Define line styles and colors for different system sizes and observables
237 line_styles = ['-', '--', ':']
238 colors = ['blue', 'green', 'red']  # Colors for sigma_x, sigma_y, sigma_z
239 observable_labels = ['x', 'y', 'z']
240
241 # System sizes
242 L_values = [6, 8, 10]
243
244 # Constants
245 W = 3  # Disorder strength
246 num_realizations = 20  # Number of disorder realizations
247
248 # Prepare the plot
249 plt.figure(figsize=(10, 6))
250 plt.title('Averaged Observable Expectation Values vs. Energy Density')
251 plt.xlabel('Energy density ($\\epsilon_n / L$)')
252 plt.ylabel(f'Expectation Value ($\\langle\\sigma_1^\\mu\\rangle_n$) with {
    num_realizations} realizations')
253
254 # Define observables
255 sigma_x = np.array([[0, 1], [1, 0]])
256 sigma_y = np.array([[0, -1j], [1j, 0]])
257 sigma_z = np.array([[1, 0], [0, -1]])
258 identity = np.identity(2)
259
260 # Loop over system sizes
261 for i, L in enumerate(L_values):
262     # Initialize arrays to store averaged expectation values for each observable
263     averaged_expectations = [np.zeros((2**L,)) for _ in range(3)]  # For sigma_x, sigma_y
    , sigma_z
264
265     # Loop over disorder realizations
266     for _ in range(num_realizations):
267         H = periodic_dense_hamiltonian_mbl(L, W)
268         eigenvalues, eigenvectors = np.linalg.eigh(H)
269
270         full_observables = [tensor_product([sigma] + [identity] * (L - 1)) for sigma in [
    sigma_x, sigma_y, sigma_z]]
271
272         # Compute expectation values for each observable
273         for j, observable in enumerate(full_observables):
274             expectation_values = []
275             for index in range(len(eigenvalues)):
276                 result = compute_observable_expectation_eigenvalue(index, observable,
    eigenvectors)
277                 expectation_values.append(result)
278
279             # Accumulate the results for averaging
280             averaged_expectations[j] += np.array(expectation_values, dtype=np.float64) /
    num_realizations
281
```

```
    # Plotting the averaged expectation values
    for j in range(3):
        plt.plot(np.array(eigenvalues) / L, averaged_expectations[j], label=f'L={L}, $\\sigma_{{1}}^{{{observable_labels[j]}}}$', color=colors[j], linestyle=line_styles[i])

# Add legend, grid, and save the plot
plt.legend()
plt.grid(True)
plt.savefig("hw3/docs/images/p4_3_1_filtered_expectations_average.png")

import matplotlib.pyplot as plt
import numpy as np
from hw3.src.p4_3.fns import periodic_dense_hamiltonian_mbl
from hw1.src.hw1 import tensor_product
from hw2.src.p5_2 import entanglement_entropy, calculate_reduced_density_matrix

# Define line styles and colors for different system sizes and observables
colors = ['blue', 'green', 'red']  # Colors for L=6, L=8, L=10

# System sizes and disorder realizations
L_values = [6, 8, 10]
num_realizations = 30  # Number of disorder realizations
W = 3  # Disorder strength

# Prepare the plot
plt.figure(figsize=(10, 6))
plt.title('Averaged Entropic Signature of Thermalization for MBL Hamiltonian')
plt.xlabel('Normalized Eigenvalue (\epsilon_n / L)')
plt.ylabel('Averaged Normalized Entanglement Entropy (S_n / L)')

# Loop over system sizes
for i, L in enumerate(L_values):
    averaged_entropies = np.zeros((2**L,))  # Initialize array to store averaged entropies

    # Loop over disorder realizations
    for _ in range(num_realizations):
        H = periodic_dense_hamiltonian_mbl(L, W)
        eigenvalues, eigenvectors = np.linalg.eigh(H)
        entropies = []

        # Compute entropies for each eigenstate
        for eigenvector in eigenvectors.T:
            rho = calculate_reduced_density_matrix(eigenvector, L, L // 2)
            entropy = entanglement_entropy(rho)
            entropies.append(entropy)

        averaged_entropies += np.array(entropies) / num_realizations

    # Plotting the averaged entropies
    plt.plot(np.array(eigenvalues) / L, averaged_entropies / L, color=colors[i], label=f'L={L}')

# Add legend, grid, and save the plot
plt.legend()
plt.grid(True)
plt.savefig("hw3/docs/images/p4_3_entropic_signature_average.png")
```

### 4.3.2 Quantum many-body scar states

This topic was first described only very recently and has attracted much interest due to its implications for our understanding of quantum chaos. The concept of a (single-particle) quantum scar state has been known for some time, and is based on the phenomenon of periodic orbits in classical chaotic systems, which can exist in principle but are unstable to small perturbations. By passing to the quantum analog of a classical chaotic system, one finds that certain eigenstates display a "scar," a nonergodic signature similar to the classical case. However, such a pattern was not expected beyond single-particle dynamics; in many-body systems at high energy, a quantum scar state would provide a sharp contrast between the validities of strong and weak ETH.

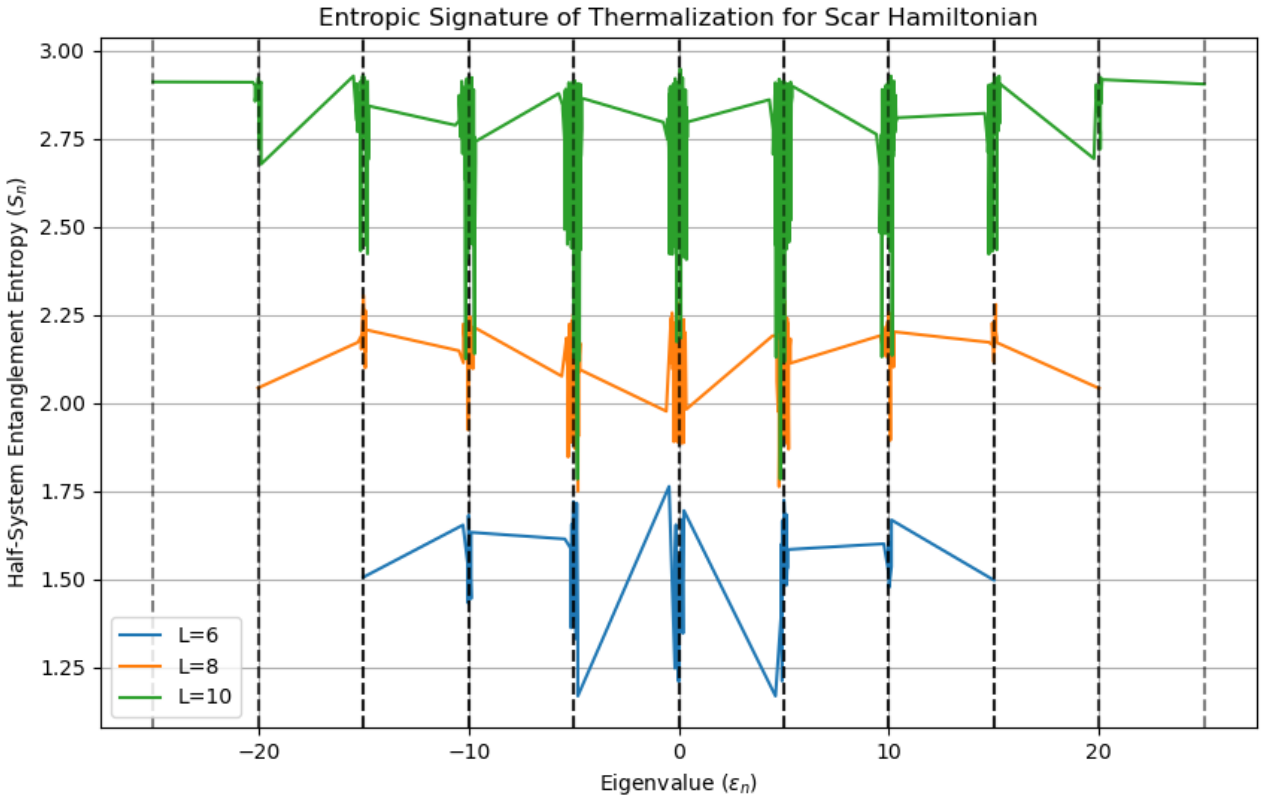
Therefore, it was surprising when such states were apparently observed in experiments with cold Rydberg atoms. While the Hamiltonian for these systems is tricky due to a complicated structure in the Hilbert space, we can instead study a toy model, the following spin-1/2 Hamiltonian [1):

$$H = \frac{\Omega}{2} \sum_{j=1}^{L} \sigma_j^x + \sum_{j=1}^{L} P_{j,j+1} \sigma_{j+2}^z \tag{12}$$

where $P_{j,j+1} = (1 - \boldsymbol{\sigma}_j \cdot \boldsymbol{\sigma}_{j+1})/4$ projects onto the singlet subspace of sites $j$ and $j+1$. Note that the mechanisms of avoided thermalization are absent: (12) is neither disordered nor integrable (the latter can be seen from the Jordan-Wigner transformation). Thus one expects ETH to apply.

In order to fully break the internal spin-rotation symmetry, $H$ contains a three-site term which at first may seem tedious to code. However, recall that $\sigma_{j+2}^z$ is diagonal in the standard basis for the many-body Hilbert space, and its effect is to contribute a sign to the matrix elements of the two-body term $P_{j,j+1}$. Thus, you can focus on the nonzero matrix elements of only the two-body operator, and afterward simply add a sign based on the parity of spin $j+2$.

In order to observe the scar states, you should compute the half-system entanglement entropy for the entire spectrum as in Sec. 4.2.2, however now you will observe several states which are
evidently far less entangled than the typical eigenstate. For this simple model, the number and exact energies of the scar states are known, indexed by $m = \{-L/2, -L/2+1, \ldots, L/2-1, L/2\}$ (that is, the $S^z$ spin states of an overall spin $s = L/2$ system), with harmonically spaced energies $E_m = \Omega m$. Indicate these energy values on your plot.




```python
def periodic_dense_hamiltonian_scars(L, omega):
    # Initialize Hamiltonian to zero matrix
    H = np.zeros((2 ** L, 2 ** L), dtype=np.complex128)

    # Define Pauli matrices
    sigma_x = np.array([[0, 1], [1, 0]])
    sigma_y = np.array([[0, -1j], [1j, 0]])
    sigma_z = np.array([[1, 0], [0, -1]])
    I = np.identity(2)

    # Tensor product helper function
    def tensor_product(matrices):
        """Calculate the tensor product of a list of matrices."""
        result = matrices[0]
        for matrix in matrices[1:]:
            result = np.kron(result, matrix)
        return result

    # Transverse field term
    for j in range(L):
        matrices = [I] * L
        matrices[j] = sigma_x
        H += omega / 2 * tensor_product(matrices)

    # Interaction term with periodic boundary conditions
    for j in range(L):
        # Ensure periodic boundary conditions
        jp1 = (j + 1) % L  # j+1 with periodic boundary
```

```
        jp2 = (j + 2) % L  # j+2 with periodic boundary

        # Projector onto the singlet state for spins j and j+1
        P = 0.25 * (np.kron(I, I) - np.kron(sigma_x, sigma_x) - np.kron(sigma_y, sigma_y)
    - np.kron(sigma_z, sigma_z))

        # Full interaction term, P_{j, j+1} \sigma_{j+2}^z
        matrices = [I] * L
        matrices[j] = P[0:2, 0:2]   # Top left block of P
        matrices[jp1] = P[2:4, 2:4] # Bottom right block of P
        matrices[jp2] = sigma_z
        H += tensor_product(matrices)

    return H

import numpy as np
import matplotlib.pyplot as plt
from hw3.src.p4_3.fns import periodic_dense_hamiltonian_scars
from hw2.src.p5_2 import entanglement_entropy, calculate_reduced_density_matrix

# Define system parameters
L_values = [6, 8, 10]
omega = 5

# Initialize the plotting environment outside the loop
plt.figure(figsize=(10, 6))
plt.title("Entropic Signature of Thermalization for Scar Hamiltonian")
plt.xlabel(f'Eigenvalue ($\epsilon_n$)')
plt.ylabel(f'Half-System Entanglement Entropy ($S_n$)')

# Generate and analyze Hamiltonians for each system size
for L in L_values:
    # Generate the Hamiltonian
    H = periodic_dense_hamiltonian_scars(L, omega)
    # Diagonalize the Hamiltonian
    eigenvalues, eigenvectors = np.linalg.eigh(H)

    # Compute entanglement entropy for each eigenstate
    entropies = [entanglement_entropy(calculate_reduced_density_matrix(vec, L, L // 2))
    for vec in eigenvectors]

    # Plot entanglement entropy
    plt.plot(np.array(eigenvalues), np.array(entropies), label=f'L={L}')

    # annotate the energy valdes of the scars on the plot; that is mark evidently far
    less entangled than the typical eigenstate. For this simple model, the number and
    exact energies of the scar states are known, indexed by $m=\{-L / 2,-L / 2+1, \ldots,
     L / 2-1, L / 2\}$ (that is, the $S^{z}$ spin states of an overall spin $s=L / 2$
    system), with harmonically spaced energies $E_{m}=\Omega m$. Indicate these energy
    values on your plot.
    for m in range(-L // 2, L // 2 + 1):
        plt.axvline(x=m * omega, color='black', linestyle='--', alpha=0.5)

# Add legend and grid to the plot
plt.legend()
plt.grid(True)
plt.savefig("hw3/docs/images/p4_3_scars_entropic_signature.png")
```

Optional. Though the scar states are relatively unentangled, they still may display volume-law scaling with a smaller prefactor. Using several system sizes, try to determine the scaling with $L$ of the scar state entanglement entropy. Do you find evidence for an area law, volume law, or logarithmic scaling?

### 4.3.3 Optional. Integrable Ising model and GGE

An integrable model can be obtained by setting $h^z = 0$ in the translation invariant Hamiltonian (8), recovering the transverse-field quantum Ising model. This model will not equilibrate to a thermal state, but it has been proposed that such a system does reach a state that is a natural generalization, obtained by taking the conservation laws into account. This equilibrium state is known as the generalized Gibbs ensemble (GGE). As this is a somewhat more technical area, we will not go into more detail regarding the GGE, but you may make direct comparisons with the previous cases by again repeating the dynamical and eigenstate experiments of Secs. 4.1 and 4.2 . You may also look into existing numerical results on the GGE to gain intuition about what to expect in this case.

## References

[1] Choi, Soonwon, Christopher J. Turner, Hannes Pichler, Wen Wei Ho, Alexios A. Michailidis, Zlatko Papić, Maksym Serbyn, Mikhail D. Lukin, and Dmitry A. Abanin. "Emergent SU(2) dynamics and perfect quantum many-body scars." Physical review letters 122, no. 22 (2019): 220603.