Quantum State Evolution for Permutation Symmetric Hamiltonians

Complete Workflow: Construction, Evolution, and Partial Trace

Research Documentation

October 31, 2025

Abstract

This document provides comprehensive documentation for the complete quantum state evolution workflow designed for research with permutation symmetric Hamiltonians. The program handles: (1) Construction of symmetric Hamiltonians for N identical qubits, (2) Evolution of quantum states using the matrix exponential e^{-iHt}, (3) Computation of density matrices, and (4) Extraction of reduced states via partial trace. This workflow is essential for mean-field quantum computing and many-body quantum systems research.

Contents

1 Introduction

1.1 Research Context

In quantum many-body systems with N identical particles, **permutation symmetry** greatly simplifies the analysis. A Hamiltonian is permutation symmetric if it treats all particles identically:

$$H = \sum_i h_i + \sum_{i<j} V_{ij} \tag{1}$$

where h_i and V_{ij} are the same for all i and all pairs (i, j).

1.2 The Complete Workflow

This program implements the full research pipeline:

1. **Input**: Hamiltonian H, initial state $|\psi(0)\rangle$, time t, number of copies N
2. **Evolution**: Compute $|\psi(t)\rangle = e^{-iHt} |\psi(0)\rangle$
3. **Density Matrix**: Form $\rho(t) = |\psi(t)\rangle \langle\psi(t)|$
4. **Partial Trace**: Extract reduced state $\rho_{\text{reduced}} = \text{Tr}_{\text{other}}[\rho(t)]$
5. **Output**: Single-copy or few-copy quantum state

2 Mathematical Foundation

2.1 Time Evolution

The Schrödinger equation gives:

$$|\psi(t)\rangle = e^{-iHt} |\psi(0)\rangle \tag{2}$$

For diagonalizable $H = UDU^\dagger$:

$$e^{-iHt} = Ue^{-iDt}U^\dagger = U \begin{pmatrix} e^{-i\lambda_1 t} & 0 & \cdots \\ 0 & e^{-i\lambda_2 t} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} U^\dagger \tag{3}$$

2.2 Density Matrix

For a pure state $|\psi\rangle$:

$$\rho = |\psi\rangle \langle\psi| \tag{4}$$

Properties:

- Hermitian: $\rho = \rho^\dagger$
- Trace one: $\text{Tr}(\rho) = 1$
- Positive semi-definite: eigenvalues ≥ 0
- Purity: $\text{Tr}(\rho^2) = 1$ for pure states, < 1 for mixed

2.3 Partial Trace

For a composite system $\mathcal{H}_A \otimes \mathcal{H}_B$ with density matrix ρ_{AB}, the reduced state of subsystem A is:

$$\rho_A = \mathrm{Tr}_B[\rho_{AB}] = \sum_i (\mathbb{I}_A \otimes \langle i|_B)\rho_{AB}(\mathbb{I}_A \otimes |i\rangle_B) \tag{5}$$

where $\{|i\rangle_B\}$ is a basis for subsystem B.

2.3.1 Example: 2-Qubit System

For $\rho = |\psi\rangle\langle\psi|$ where $|\psi\rangle = \sum_{ij} c_{ij} |i\rangle |j\rangle$:

$$\rho_A = \mathrm{Tr}_B[\rho] = \sum_k (\mathbb{I} \otimes \langle k|)\rho(\mathbb{I} \otimes |k\rangle) \tag{6}$$

In matrix form:

$$(\rho_A)_{ij} = \sum_k \rho_{(i,k)(j,k)} \tag{7}$$

3 Program Structure

The program is organized into 8 main parts:

1. **Hamiltonian Construction**: Build symmetric Hamiltonians
2. **State Evolution**: Compute $e^{-iHt}|\psi(0)\rangle$
3. **Density Matrix**: Form $\rho = |\psi\rangle\langle\psi|$
4. **Partial Trace**: Extract reduced states
5. **Complete Workflow**: Integrated pipeline function
6. **Pre-built Hamiltonians**: Common models (TFIM, XY, etc.)
7. **Utility Functions**: Product states, Bloch vectors
8. **Demonstration**: Working examples

4 Usage Guide

4.1 Quick Start

The main function is `evolve_and_trace()`:

```python
from quantum_state_evolution_complete import *

# Step 1: Construct Hamiltonian
N = 4  # Number of qubits
H = construct_TFIM(N, J=1.0, h=0.5)

# Step 2: Create initial state
psi0 = create_product_state(N, [1, 0])  # |0000>

# Step 3: Evolve and extract single-qubit state
t = 1.0
```

```python
results = evolve_and_trace(H, psi0, t, N, keep_qubits=[0])

# Step 4: Access results
rho_1 = results['rho_reduced']  # Single-qubit density matrix
print(rho_1)
```

Listing 1: Basic usage

4.2 Main Function: evolve_and_trace()

```python
results = evolve_and_trace(
    H,                      # Hamiltonian (2^N x 2^N array)
    psi0,                  # Initial state (2^N array)
    t,                     # Evolution time (float)
    N,                     # Number of qubits (int)
    keep_qubits=[0],       # Qubits to keep (list)
    method='auto',         # Evolution method
    verbose=True           # Print progress
)
```

Listing 2: Complete function signature

4.2.1 Parameters

- **H**: Permutation symmetric Hamiltonian matrix, shape $(2^N, 2^N)$
- **psi0**: Initial quantum state vector, shape $(2^N,)$
- **t**: Evolution time (float, in natural units $\hbar = 1$)
- **N**: Total number of qubits (int)
- **keep_qubits**: List of qubit indices to keep after tracing
  - `[0]`: Keep first qubit only
  - `[0, 1]`: Keep first two qubits
  - `[2]`: Keep third qubit
- **method**: Evolution method
  - `'auto'`: Automatic (default)
  - `'diagonalize'`: Use eigendecomposition
  - `'expm'`: Use matrix exponential
- **verbose**: Print progress information (bool)

4.2.2 Returns

Dictionary with keys:

- `'psi_0'`: Initial state
- `'psi_t'`: Evolved state $|\psi(t)\rangle$
- `'rho_full'`: Full density matrix $\rho(t)$

- `'rho_reduced'`: Reduced density matrix after partial trace
- `'evolution_info'`: Evolution information
- `'density_info'`: Density matrix diagnostics

5 Constructing Hamiltonians

5.1 Pre-built Models

5.1.1 Transverse Field Ising Model (TFIM)

$$H = -J \sum_{i<j} Z_i \otimes Z_j - h \sum_i X_i \tag{8}$$

```
H = construct_TFIM(N=4, J=1.0, h=0.5)
```

Listing 3: TFIM construction

Parameters:

- J: Ising coupling strength (typically 0.5–2.0)
- h: Transverse field strength (typically 0.1–1.0)

Physics: Quantum annealing, phase transitions, optimization

5.1.2 XY Model

$$H = J \sum_{i<j} X_i \otimes X_j \tag{9}$$

```
H = construct_XY_model(N=4, J=0.5)
```

Listing 4: XY model construction

Physics: State transfer, entanglement generation

5.1.3 General Mean-Field Model

$$H = \sum_i (h_x X_i + h_z Z_i) + \sum_{i<j} (J_{xx} X_i \otimes X_j + J_{zz} Z_i \otimes Z_j) \tag{10}$$

```
H = construct_general_MF(
    N=4,
    h_x=0.3,     # Transverse field
    h_z=0.1,     # Longitudinal field
    J_xx=0.2,    # XX coupling
    J_zz=0.4     # ZZ coupling
)
```

Listing 5: General model construction

5.2 Custom Hamiltonians

Build custom symmetric Hamiltonians using building blocks:

```
N = 6
H_custom = np.zeros((2**N, 2**N), dtype=complex)

# Add single-qubit terms
H_custom += add_single_X(N, coefficient=0.5)
H_custom += add_single_Z(N, coefficient=0.2)

# Add two-qubit interactions
H_custom += add_ZZ_interaction(N, J=1.0)
H_custom += add_XX_interaction(N, J=0.3)
```

Listing 6: Custom Hamiltonian

Available building blocks:

- `add_single_X(N, coeff)`: $\sum_i X_i$
- `add_single_Z(N, coeff)`: $\sum_i Z_i$
- `add_XX_interaction(N, J)`: $\sum_{i<j} X_i \otimes X_j$
- `add_ZZ_interaction(N, J)`: $\sum_{i<j} Z_i \otimes Z_j$

6 Creating Initial States

6.1 Product States

Create $|\psi\rangle^{\otimes N}$ from single-qubit state $|\psi\rangle$:

```
# |0>^ N  (all qubits in |0>)
psi0 = create_product_state(N, [1, 0])

# |1>^ N  (all qubits in |1>)
psi0 = create_product_state(N, [0, 1])

# |+>^ N  (all qubits in superposition)
psi0 = create_product_state(N, [1, 1]/np.sqrt(2))

# Custom single-qubit state
alpha, beta = 0.8, 0.6
psi0 = create_product_state(N, [alpha, beta])
```

Listing 7: Product state creation

6.2 Custom States

For non-product states, construct manually:

```
# GHZ-type state
N = 3
psi0 = np.zeros(2**N)
psi0[0] = 1/np.sqrt(2)   # |000>
psi0[-1] = 1/np.sqrt(2)  # |111>

# W state
psi0 = np.zeros(8)
```

```
psi0[0b001] = 1/np.sqrt(3)
psi0[0b010] = 1/np.sqrt(3)
psi0[0b100] = 1/np.sqrt(3)
```

Listing 8: Custom initial state

7 Complete Examples

7.1 Example 1: Single-Qubit Extraction

Task: Evolve 4-qubit system and extract single-qubit state.

```python
import numpy as np
from quantum_state_evolution_complete import *

# Setup
N = 4
H = construct_TFIM(N, J=1.0, h=0.5)
psi0 = create_product_state(N, [1, 0])  # |0000>
t = 1.0

# Execute workflow
results = evolve_and_trace(H, psi0, t, N, keep_qubits=[0])

# Extract results
rho_1 = results['rho_reduced']  # 2x2 matrix
purity = results['density_info']['reduced_purity']

# Compute Bloch vector
r = bloch_vector(rho_1)

print(f"Single-qubit density matrix:\n{rho_1}")
print(f"Bloch vector: {r}")
print(f"Purity: {purity:.4f}")
```

Listing 9: Single-qubit extraction

Output:

```
Single-qubit density matrix:
[[0.9907 +0.j      0.0288+0.0248j]
 [0.0288-0.0248j  0.0093+0.j     ]]
Bloch vector: [0.0576 0.0495 0.9814]
Purity: 0.9845
```

7.2 Example 2: Two-Qubit Extraction

Task: Extract two-qubit reduced state.

```python
# Keep first two qubits
results = evolve_and_trace(
    H, psi0, t, N,
    keep_qubits=[0, 1]  # Keep qubits 0 and 1
)

rho_12 = results['rho_reduced']  # 4x4 matrix
print(f"Shape: {rho_12.shape}")
print(f"Trace: {np.trace(rho_12)}")
```

Listing 10: Two-qubit extraction

7.3 Example 3: Time Evolution Sequence

Task: Track evolution over multiple time steps.

```
N = 4
H = construct_TFIM(N, J=1.0, h=0.5)
psi0 = create_product_state(N, [1, 1]/np.sqrt(2))

time_points = np.linspace(0, 5, 50)
bloch_vectors = []

for t in time_points:
    results = evolve_and_trace(
        H, psi0, t, N,
        keep_qubits=[0],
        verbose=False
    )
    rho_1 = results['rho_reduced']
    r = bloch_vector(rho_1)
    bloch_vectors.append(r)

bloch_vectors = np.array(bloch_vectors)

# Plot trajectory
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(bloch_vectors[:, 0],
        bloch_vectors[:, 1],
        bloch_vectors[:, 2])
ax.set_xlabel('$r_x$')
ax.set_ylabel('$r_y$')
ax.set_zlabel('$r_z$')
plt.title('Bloch␣Vector␣Trajectory')
plt.show()
```

Listing 11: Time evolution sequence

7.4 Example 4: Parameter Sweep

Task: Compare different Hamiltonian parameters.

```
N = 4
psi0 = create_product_state(N, [1, 0])
t = 1.0

# Sweep transverse field
h_values = np.linspace(0.1, 2.0, 20)
purities = []

for h in h_values:
    H = construct_TFIM(N, J=1.0, h=h)
    results = evolve_and_trace(
        H, psi0, t, N,
        keep_qubits=[0],
        verbose=False
    )
    purity = results['density_info']['reduced_purity']
    purities.append(purity)

# Plot
plt.plot(h_values, purities)
plt.xlabel('Transverse␣field␣h')
plt.ylabel('Single-qubit␣purity')
plt.title('Purity␣vs␣Transverse␣Field')
plt.show()
```

Listing 12: Parameter sweep

8 Understanding the Results

8.1 Reduced Density Matrix

The reduced density matrix ρ_{reduced} characterizes the quantum state of the kept qubits after tracing out the others.

For single qubit (2×2 matrix):

$$\rho_1 = \begin{pmatrix} \rho_{00} & \rho_{01} \\ \rho_{10} & \rho_{11} \end{pmatrix} \tag{11}$$

where:

- ρ_{00}: Probability of being in $|0\rangle$
- ρ_{11}: Probability of being in $|1\rangle$
- ρ_{01}, ρ_{10}: Coherences (quantum correlations)

8.2 Bloch Vector

For single-qubit states, the Bloch vector $\vec{r} = (r_x, r_y, r_z)$ provides geometric intuition:

$$\rho = \frac{1}{2}(I + \vec{r} \cdot \vec{\sigma}) \tag{12}$$

where $\vec{\sigma} = (X, Y, Z)$ are Pauli matrices.

Properties:

- $|\vec{r}| = 1$: Pure state
- $|\vec{r}| < 1$: Mixed state (entangled with environment)
- $\vec{r} = (0, 0, 1)$: State $|0\rangle$
- $\vec{r} = (0, 0, -1)$: State $|1\rangle$
- $\vec{r} = (1, 0, 0)$: State $|+\rangle$

8.3 Purity

Purity measures how "pure" vs "mixed" a state is:

$$\mathcal{P} = \mathrm{Tr}(\rho^2) \tag{13}$$

Values:

- $\mathcal{P} = 1$: Pure state (no entanglement with environment)
- $0 < \mathcal{P} < 1$: Mixed state (entangled or classically mixed)
- $\mathcal{P} = 1/d$: Maximally mixed state (d = dimension)

For single qubit: $\mathcal{P} = |\vec{r}|^2$

9 Practical Considerations

9.1 Computational Complexity

Operation	**Complexity**	**Bottleneck**
Hamiltonian construction	$O(N^2 \cdot 2^N)$	Pairwise interactions
State evolution (diagonalize)	$O(2^{3N})$	Eigendecomposition
Density matrix	$O(2^{2N})$	Outer product
Partial trace	$O(2^{2N})$	Summing over basis

Table 1: Computational complexity

Practical limits:

- $N \leq 10$: Feasible on laptop
- $N = 11 - 14$: Requires workstation
- $N > 14$: Needs HPC or approximations

9.2 Numerical Accuracy

Check these diagnostics:

```python
results = evolve_and_trace(H, psi0, t, N, keep_qubits=[0])

# Evolution unitarity
print(f"Unitarity error: {results['evolution_info']['unitarity_error']}")

# Density matrix validity
print(f"Trace error: {results['density_info']['reduced_errors']['trace_error']}")
print(f"Hermiticity error: {results['density_info']['reduced_errors']['hermitian_error']}")

# Typical values:
# Unitarity error < 1e-12 (good)
# Trace error < 1e-10 (good)
# Hermiticity error < 1e-10 (good)
```

Listing 13: Accuracy checks

9.3 Common Issues

9.3.1 Non-Hermitian Hamiltonian

Symptom: Warning "Hamiltonian is not Hermitian"
Cause: $H \neq H^\dagger$
Fix:

```
# Symmetrize
H = (H + H.conj().T) / 2
```

9.3.2 Trace Not Equal to 1

Symptom: $\mathrm{Tr}(\rho) \neq 1$
Cause: Numerical error accumulation
Fix: Reduce time step or use higher precision

9.3.3 Purity ¿ 1

Symptom: $\mathrm{Tr}(\rho^2) > 1$
Cause: Numerical errors
Fix: Check Hamiltonian construction and normalization

10 Advanced Usage

10.1 Using Custom Evolution Method

Force specific evolution method:

```
# Use diagonalization (exact)
results = evolve_and_trace(H, psi0, t, N,
                           keep_qubits=[0],
                           method='diagonalize')

# Use matrix exponential (general)
results = evolve_and_trace(H, psi0, t, N,
                           keep_qubits=[0],
                           method='expm')
```

10.2 Extracting Multiple Observables

```
results = evolve_and_trace(H, psi0, t, N, keep_qubits=[0])

rho_1 = results['rho_reduced']

# Bloch vector
r = bloch_vector(rho_1)

# Expectation values
exp_X = 2 * np.real(rho_1[0, 1])
exp_Y = 2 * np.imag(rho_1[1, 0])
exp_Z = np.real(rho_1[0, 0] - rho_1[1, 1])

# Populations
P_0 = np.real(rho_1[0, 0])
P_1 = np.real(rho_1[1, 1])
```

```python

print(f"<X> = {exp_X:.4f}")
print(f"<Y> = {exp_Y:.4f}")
print(f"<Z> = {exp_Z:.4f}")
print(f"P(0) = {P_0:.4f}, P(1) = {P_1:.4f}")
```

11 Validation and Testing

11.1 Test Case 1: Known Evolution

For $H = \omega Z$, $|\psi(0)\rangle = |+\rangle$:

$$|\psi(t)\rangle = \frac{1}{\sqrt{2}}(e^{-i\omega t}|0\rangle + e^{i\omega t}|1\rangle) \tag{14}$$

11.2 Test Case 2: Conservation Laws

For any Hamiltonian:

- Norm preserved: $\| |\psi(t)\rangle \| = \| |\psi(0)\rangle \|$
- Trace preserved: $\mathrm{Tr}(\rho(t)) = 1$

11.3 Test Case 3: Permutation Symmetry

For symmetric H and product initial state:

$$\rho_i = \rho_j \quad \forall i, j \tag{15}$$

All single-qubit reduced states should be identical.

12 Troubleshooting

Problem	Possible Cause	Solution
Memory error	System too large ($N > 12$)	Reduce N or use approximations
Slow execution	Large N ($N > 10$)	Use sparse methods or reduce N
Non-unitary evolution	Non-Hermitian H	Check $H = H^\dagger$
Wrong dimensions	Index mismatch	Check N, `keep_qubits`

Table 2: Common problems and solutions

13 References

13.1 Key Equations Summary

$$\text{Evolution:} \quad |\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle \tag{16}$$

$$\text{Density matrix:} \quad \rho = |\psi\rangle\langle\psi| \tag{17}$$

$$\text{Partial trace:} \quad \rho_A = \mathrm{Tr}_B[\rho_{AB}] \tag{18}$$

$$\text{Purity:} \quad \mathcal{P} = \mathrm{Tr}(\rho^2) \tag{19}$$

$$\text{Bloch vector:} \quad \rho = \frac{1}{2}(I + \vec{r}\cdot\vec{\sigma}) \tag{20}$$

13.2 Further Reading

- Nielsen & Chuang, *Quantum Computation and Quantum Information*
- Breuer & Petruccione, *The Theory of Open Quantum Systems*
- Sachdev, *Quantum Phase Transitions*

14 Conclusion

This program provides a complete workflow for studying permutation symmetric quantum systems. The integrated pipeline from Hamiltonian construction to reduced state extraction enables efficient research on mean-field quantum dynamics, many-body entanglement, and quantum phase transitions.

For questions or issues, refer to the source code documentation or contact the research group.

A Function Reference

A.1 Main Functions

- `evolve_and_trace(H, psi0, t, N, keep_qubits, method, verbose)`
- `evolve_state(psi0, H, t, method)`
- `partial_trace(rho, N, keep_qubits)`
- `state_to_density_matrix(psi)`

A.2 Hamiltonian Construction

- `construct_TFIM(N, J, h)`
- `construct_XY_model(N, J)`
- `construct_general_MF(N, h_x, h_z, J_xx, J_zz)`
- `add_single_X(N, coefficient)`
- `add_single_Z(N, coefficient)`
- `add_XX_interaction(N, J)`
- `add_ZZ_interaction(N, J)`

A.3 Utilities

- `create_product_state(N, single_qubit_state)`
- `bloch_vector(rho_1)`
- `verify_density_matrix(rho, tol)`