Quantum State Evolution using e^{iHt}

User Guide and Mathematical Reference

Kyro Jeremy Gibling

October 31, 2025

Abstract

This document provides comprehensive documentation for the `quantum_evolution.py` module, which implements quantum state evolution using the matrix exponential e^{-iHt}. The module handles both diagonal and general diagonalizable Hamiltonians efficiently, with automatic detection and method selection. This guide covers the mathematical background, usage instructions, and practical examples.

Contents

1 Introduction **3**

2 Mathematical Background **3**
- 2.1 Matrix Exponential . . . 3
- 2.2 Diagonalization Method . . . 3
- 2.3 Special Case: Diagonal Hamiltonian . . . 4
- 2.4 Unitarity . . . 4

3 Implementation Methods **4**
- 3.1 Method 1: Diagonal . . . 4
- 3.2 Method 2: Diagonalize . . . 5
- 3.3 Method 3: Matrix Exponential (expm) . . . 5
- 3.4 Method Selection (Auto) . . . 5

4 Usage Guide **5**
- 4.1 Basic Usage . . . 5
- 4.2 Function Signature . . . 6
- 4.3 Parameters . . . 6
- 4.4 Return Values . . . 6

5 Examples **6**
- 5.1 Example 1: Diagonal Hamiltonian . . . 6
- 5.2 Example 2: Non-Diagonal Hamiltonian . . . 7
- 5.3 Example 3: Mean-Field Hamiltonian . . . 7
- 5.4 Example 4: Method Comparison . . . 8

6 Practical Considerations **8**
- 6.1 When to Use Each Method . . . 8
- 6.2 Numerical Accuracy . . . 8
- 6.3 Common Pitfalls . . . 8
- 6.4 Checking Results . . . 9

7 Mathematical Formulas Reference **9**
7.1 Time Evolution 9
7.2 For 2×2 Systems 9
7.3 Eigenvalue Decomposition 9

8 Troubleshooting **9**
8.1 Error Messages 9
8.2 Warnings 10

9 Performance Tips **10**

10 Conclusion **10**

A Complete API Reference **10**
A.1 Main Function 10
A.2 Utility Functions 10
A.3 Internal Functions 10

1 Introduction

In quantum mechanics, the time evolution of a quantum state $|\psi(t)\rangle$ under a time-independent Hamiltonian H is governed by the Schrödinger equation:

$$i\hbar\frac{d|\psi\rangle}{dt} = H|\psi\rangle \tag{1}$$

Setting $\hbar = 1$ (natural units), the formal solution is:

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

where e^{-iHt} is the time evolution operator.

This module provides efficient numerical methods to compute $e^{-iHt}|\psi(0)\rangle$ for various types of Hamiltonians.

2 Mathematical Background

2.1 Matrix Exponential

The matrix exponential is defined by the power series:

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots = \sum_{k=0}^{\infty}\frac{A^k}{k!} \tag{3}$$

For the time evolution operator:

$$e^{-iHt} = I - iHt + \frac{(-iHt)^2}{2!} + \frac{(-iHt)^3}{3!} + \cdots \tag{4}$$

2.2 Diagonalization Method

If H is diagonalizable, we can write:

$$H = UDU^\dagger \tag{5}$$

where:

- U is a unitary matrix whose columns are the eigenvectors of H
- $D = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is a diagonal matrix of eigenvalues
- $U^\dagger$ is the conjugate transpose of U

Then the matrix exponential becomes:

$$e^{-iHt} = Ue^{-iDt}U^\dagger \tag{6}$$

where:

$$e^{-iDt} = \begin{pmatrix} e^{-i\lambda_1 t} & 0 & \cdots & 0 \\ 0 & e^{-i\lambda_2 t} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{-i\lambda_n t} \end{pmatrix} \tag{7}$$

This reduces the problem to:

1. Finding eigenvalues and eigenvectors (diagonalization)

2. Exponentiating scalars (trivial)

3. Matrix multiplication

2.3 Special Case: Diagonal Hamiltonian

If H is already diagonal:

$$H = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix} \tag{8}$$

Then e^{-iHt} is simply:

$$e^{-iHt} = \begin{pmatrix} e^{-i\lambda_1 t} & 0 & \cdots & 0 \\ 0 & e^{-i\lambda_2 t} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{-i\lambda_n t} \end{pmatrix} \tag{9}$$

The evolved state is:

$$\psi_i(t) = e^{-i\lambda_i t}\psi_i(0) \tag{10}$$

This is the most efficient case, requiring only $O(n)$ operations.

2.4 Unitarity

For a Hermitian Hamiltonian ($H = H^\dagger$), the time evolution operator is unitary:

$$(e^{-iHt})^\dagger e^{-iHt} = I \tag{11}$$

This ensures that:

$$\langle\psi(t)|\psi(t)\rangle = \langle\psi(0)|\psi(0)\rangle = 1 \tag{12}$$

The norm is preserved: $\| \, |\psi(t)\rangle \, \| = \| \, |\psi(0)\rangle \, \|$

3 Implementation Methods

The module implements three methods for computing $e^{-iHt} \, |\psi(0)\rangle$:

3.1 Method 1: Diagonal

Use when: H is already diagonal
Algorithm:

1. Extract diagonal elements: $\lambda_i = H_{ii}$

2. Compute phases: $\phi_i = e^{-i\lambda_i t}$

3. Element-wise multiplication: $\psi_i(t) = \phi_i \cdot \psi_i(0)$

Complexity: $O(n)$

3.2 Method 2: Diagonalize

Use when: H is diagonalizable (but not diagonal)
Algorithm:

1. Diagonalize: $H = UDU^\dagger$ using eigenvalue decomposition
2. Compute diagonal exponential: e^{-iDt}
3. Transform state: $|\psi(t)\rangle = Ue^{-iDt}U^\dagger |\psi(0)\rangle$

Complexity: $O(n^3)$ for diagonalization, $O(n^2)$ for matrix-vector products

3.3 Method 3: Matrix Exponential (expm)

Use when: General case, or when high accuracy is needed
Algorithm:

1. Compute full matrix: $U = e^{-iHt}$ using Padé approximation
2. Matrix-vector product: $|\psi(t)\rangle = U |\psi(0)\rangle$

Complexity: $O(n^3)$ for matrix exponential

3.4 Method Selection (Auto)

The `'auto'` method automatically selects the best approach:

1. Check if H is diagonal (compare off-diagonal elements to tolerance)
2. If diagonal: use Method 1
3. If not diagonal: use Method 2 (diagonalize)

4 Usage Guide

4.1 Basic Usage

```python
import numpy as np
from quantum_evolution import evolve_state

# Define Hamiltonian
H = np.array([[1, 0], [0, -1]])  # Diagonal

# Define initial state
psi0 = np.array([1, 1]) / np.sqrt(2)  # |+> state

# Evolve for time t=1.0
t = 1.0
psi_t, info = evolve_state(psi0, H, t)

print(f"Evolved state: {psi_t}")
print(f"Method used: {info['method_used']}")
```

Listing 1: Basic state evolution

4.2 Function Signature

```
psi_t, info = evolve_state(
    psi0,                       # Initial state vector
    H,                          # Hamiltonian matrix
    t,                          # Evolution time
    method='auto',              # 'auto', 'diagonal', 'diagonalize', 'expm'
    check_diagonal_tol=1e-10 # Tolerance for diagonal check
)
```

Listing 2: Complete function signature

4.3 Parameters

- **psi0**: Initial state vector $|\psi(0)\rangle$ (1D numpy array, complex or real)
- **H**: Hamiltonian matrix (2D numpy array, should be Hermitian)
- **t**: Evolution time (float)
- **method**: Method selection
  - `'auto'`: Automatic detection (recommended)
  - `'diagonal'`: Assume H is diagonal
  - `'diagonalize'`: Force diagonalization
  - `'expm'`: Use scipy's matrix exponential
- **check_diagonal_tol**: Tolerance for checking if matrix is diagonal

4.4 Return Values

The function returns a tuple `(psi_t, info)` where:

- **psi_t**: Evolved state $|\psi(t)\rangle$ (numpy array)
- **info**: Dictionary containing:
  - `'method_used'`: Actual method used
  - `'is_diagonal'`: Whether H was detected as diagonal
  - `'eigenvalues'`: Eigenvalues of H (if computed)
  - `'norm_initial'`: $\| |\psi(0)\rangle \|$
  - `'norm_final'`: $\| |\psi(t)\rangle \|$
  - `'unitarity_error'`: $|\| |\psi(t)\rangle \| - \| |\psi(0)\rangle \||$

5 Examples

5.1 Example 1: Diagonal Hamiltonian

Evolution under $H = \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$:

```python
import numpy as np
from quantum_evolution import evolve_state

# Pauli Z matrix
H = np.array([[1, 0], [0, -1]])

# Initial state: |+> = (|0> + |1>)/sqrt(2)
psi0 = np.array([1, 1]) / np.sqrt(2)

# Evolve
t = 1.0
psi_t, info = evolve_state(psi0, H, t)

print(f"|psi(t)> = {psi_t}")
print(f"Method: {info['method_used']}")  # Should be 'diagonal'
```

Listing 3: Diagonal Hamiltonian example

Expected result:

$$|\psi(t)\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} e^{-it} \\ e^{it} \end{pmatrix} \tag{13}$$

5.2 Example 2: Non-Diagonal Hamiltonian

Evolution under Pauli X: $H = \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$:

```python
# Pauli X matrix
H = np.array([[0, 1], [1, 0]])

# Initial state: |0>
psi0 = np.array([1, 0])

# Evolve for t = pi/2
t = np.pi / 2
psi_t, info = evolve_state(psi0, H, t)

print(f"|psi(t)> = {psi_t}")
print(f"Eigenvalues: {info['eigenvalues']}")
```

Listing 4: Non-diagonal Hamiltonian

Expected result: For $t = \pi/2$, $|0\rangle \to -i\,|1\rangle$

5.3 Example 3: Mean-Field Hamiltonian

For a mean-field Hamiltonian $H = h_x\sigma_x + h_z\sigma_z$:

```python
# Define parameters
h_x = 0.5
h_z = 1.0

# Construct Hamiltonian
X = np.array([[0, 1], [1, 0]])
Z = np.array([[1, 0], [0, -1]])
H = h_x * X + h_z * Z

# Initial state
psi0 = np.array([1, 1]) / np.sqrt(2)
```

```
12
13 # Evolve
14 t = 2.0
15 psi_t, info = evolve_state(psi0, H, t)
16
17 print(f"|psi(t)> = {psi_t}")
18 print(f"Eigenvalues: {info['eigenvalues']}")
```

Listing 5: Mean-field Hamiltonian

5.4 Example 4: Method Comparison

Compare different methods for the same evolution:

```
1 H = np.array([[1, 0.5], [0.5, -1]])
2 psi0 = np.array([1, 0])
3 t = 1.0
4
5 # Try different methods
6 for method in ['auto', 'diagonalize', 'expm']:
7     psi_t, info = evolve_state(psi0, H, t, method=method)
8     print(f"Method '{method}': |psi(t)> = {psi_t}")
```

Listing 6: Comparing methods

6 Practical Considerations

6.1 When to Use Each Method

Scenario	Method	Reason
H is diagonal	`'diagonal'`	Fastest ($O(n)$)
H is small ($n < 100$)	`'auto'` or `'diagonalize'`	Exact, efficient
H is large, sparse	`'expm'`	Exploits sparsity
Need eigenvalues	`'diagonalize'`	Returns eigenvalues
Uncertain	`'auto'`	Safe default

Table 1: Method selection guide

6.2 Numerical Accuracy

- **Diagonal method**: Exact (within machine precision)
- **Diagonalize method**: Accurate for well-conditioned matrices
- **Expm method**: High accuracy (Padé approximation with scaling and squaring)

6.3 Common Pitfalls

1. **Non-Hermitian Hamiltonian**: Results in non-unitary evolution (norm not preserved)
2. **Dimension mismatch**: Ensure $|\psi(0)\rangle$ has same dimension as H
3. **Very large eigenvalues**: May require smaller time steps for accuracy
4. **Nearly degenerate eigenvalues**: Can cause numerical issues in diagonalization

6.4 Checking Results

Always verify:

```
# Check Hermiticity
from quantum_evolution import check_hermiticity
is_herm, error = check_hermiticity(H)
print(f"Is Hermitian: {is_herm}, error: {error}")

# Check unitarity
from quantum_evolution import verify_unitarity
is_unit, error = verify_unitarity(psi0, psi_t)
print(f"Is unitary: {is_unit}, error: {error}")

# Check from info dict
print(f"Unitarity error: {info['unitarity_error']}")
```

Listing 7: Verification

7 Mathematical Formulas Reference

7.1 Time Evolution

$$|\psi(t)\rangle = e^{-iHt}\,|\psi(0)\rangle \tag{14}$$

7.2 For 2×2 Systems

For $H = h_x\sigma_x + h_z\sigma_z$, define $\Omega = \sqrt{h_x^2 + h_z^2}$:

$$e^{-iHt} = \cos(\Omega t)I - i\sin(\Omega t)\frac{H}{\Omega} \tag{15}$$

Explicitly:

$$e^{-iHt} = \begin{pmatrix} \cos(\Omega t) - i\frac{h_z}{\Omega}\sin(\Omega t) & -i\frac{h_x}{\Omega}\sin(\Omega t) \\ -i\frac{h_x}{\Omega}\sin(\Omega t) & \cos(\Omega t) + i\frac{h_z}{\Omega}\sin(\Omega t) \end{pmatrix} \tag{16}$$

7.3 Eigenvalue Decomposition

For 2×2 Hermitian matrix:

$$H = \begin{pmatrix} a & b \\ b^* & c \end{pmatrix} \tag{17}$$

Eigenvalues:

$$\lambda_\pm = \frac{a+c}{2} \pm \sqrt{\left(\frac{a-c}{2}\right)^2 + |b|^2} \tag{18}$$

8 Troubleshooting

8.1 Error Messages

- **"psi0 must be a 1D array"**: Ensure initial state is a vector, not a matrix
- **"H must be a square matrix"**: Check Hamiltonian dimensions
- **"Dimension mismatch"**: Ensure $\text{len}(\psi_0) = H.\text{shape}[0]$

- **"Hamiltonian is not Hermitian"**: Warning only; check if $H = H^\dagger$

8.2 Warnings

- **"Eigenvalues have imaginary parts"**: H may not be Hermitian
- **"Hamiltonian is not Hermitian"**: Evolution may not preserve norm

9 Performance Tips

1. For diagonal H, explicitly use `method='diagonal'` to skip detection
2. For repeated evolution with same H, diagonalize once and reuse eigenvalues
3. For very large systems, consider using sparse matrix methods
4. For many time steps, consider using ODE solvers (Runge-Kutta) instead

10 Conclusion

This module provides a flexible and efficient implementation of quantum state evolution using the matrix exponential e^{-iHt}. The automatic method selection makes it easy to use, while the specialized methods allow for optimization when needed.

For questions or issues, please refer to the source code comments or contact the author.

A Complete API Reference

A.1 Main Function

`evolve_state(psi0, H, t, method='auto', check_diagonal_tol=1e-10)`
Returns: `(psi_t, info)`

A.2 Utility Functions

- `check_hermiticity(H, tol=1e-10)`: Check if matrix is Hermitian
- `verify_unitarity(psi0, psi_t, tol=1e-10)`: Verify norm preservation

A.3 Internal Functions

- `_is_diagonal(H, tol)`: Check if matrix is diagonal
- `_evolve_diagonal(psi0, H, t)`: Evolve with diagonal H
- `_evolve_via_diagonalization(psi0, H, t)`: Evolve via eigendecomposition
- `_evolve_via_expm(psi0, H, t)`: Evolve using scipy's expm