Quantum Hamiltonian Builder & Time Evolution

Quick Reference Guide

1 Overview

This reference guide covers two main modules:

- `hamiltonian_builder.py` – Construct Hamiltonians from Pauli operators
- `runge_kutta_hamiltonian.py` – Time evolution using 4th-order Runge-Kutta

2 Hamiltonian Construction

2.1 Available Pauli Operators

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$
$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

2.2 Method 1: String Notation

Function: `hamiltonian_from_string(hamiltonian_str)`

Usage: Directly write Hamiltonian as a string using $\otimes$ (or $*$ or concatenation).

```python
from hamiltonian_builder import hamiltonian_from_string

# Example 1: Single tensor product
H = hamiltonian_from_string("X@Z")  # X tensor Z

# Example 2: Sum of terms
H = hamiltonian_from_string("X@I + I@X")

# Example 3: With coefficients
H = hamiltonian_from_string("0.5*X@Z + 0.3*Z@X")

# Example 4: Four qubits (as requested)
H = hamiltonian_from_string("Z@I@X@Z + X@X@I@I")

# Example 5: Compact notation (no symbols)
H = hamiltonian_from_string("XII + IXI + IIX")
```

Mathematical Correspondence:

$$\texttt{"X@Z"} \to X \otimes Z$$
$$\texttt{"X@I + I@X"} \to X \otimes I + I \otimes X$$
$$\texttt{"0.5*X@Z"} \to 0.5 \cdot (X \otimes Z)$$

2.3 Method 2: HamiltonianBuilder Class

Class: `HamiltonianBuilder()`

Methods:

- `add_term(operators, coefficient)` – Add a term to the Hamiltonian
- `build()` – Return the final Hamiltonian matrix

Basic Usage:

```
from hamiltonian_builder import HamiltonianBuilder

# Create builder instance
builder = HamiltonianBuilder()

# Add terms
builder.add_term(['X', 'I', 'Z'], coefficient=1.0)
builder.add_term(['Z', 'X', 'I'], coefficient=1.0)

# Build the matrix
H = builder.build()
```

Mathematical Correspondence:

`add_term(['X', 'I', 'Z'], coefficient=1.0)` $\rightarrow H = H + 1.0 \cdot (X \otimes I \otimes Z)$

`add_term(['Z', 'X', 'I'], coefficient=1.0)` $\rightarrow H = H + 1.0 \cdot (Z \otimes X \otimes I)$

Final result: $H = X \otimes I \otimes Z + Z \otimes X \otimes I$

2.4 Systematic Construction Example

For nearest-neighbor interactions: $H = \sum_{i=0}^{n-2} X_i \otimes X_{i+1}$

```
n_qubits = 4
builder = HamiltonianBuilder()

for i in range(n_qubits - 1):
    ops = ['I'] * n_qubits   # Start with all identities
    ops[i] = 'X'             # Place X at position i
    ops[i+1] = 'X'           # Place X at position i+1
    builder.add_term(ops, coefficient=1.0)

H = builder.build()
```

This constructs:

$$H = X \otimes X \otimes I \otimes I + I \otimes X \otimes X \otimes I + I \otimes I \otimes X \otimes X$$

3 Time Evolution

3.1 Schrödinger Equation

The time-dependent Schrödinger equation:

$$i\hbar \frac{d}{dt}|\psi(t)\rangle = H|\psi(t)\rangle$$

In natural units ($\hbar = 1$):

$$\frac{d}{dt}|\psi(t)\rangle = -iH|\psi(t)\rangle$$

3.2 Runge-Kutta Method

Function: `runge_kutta_4(psi0, H, t, dt=0.001)`

Parameters:

- `psi0` – Initial state vector $|\psi(0)\rangle$ (complex numpy array)
- `H` – Hamiltonian matrix (complex numpy array)
- `t` – Final time t
- `dt` – Time step (default: 0.001)

Returns: Evolved state $|\psi(t)\rangle$

Usage:

```
import numpy as np
from runge_kutta_hamiltonian import runge_kutta_4

# Define Hamiltonian
H = hamiltonian_from_string("X@Z")

# Initial state |00> (two qubits)
psi0 = np.array([1, 0, 0, 0], dtype=complex)

# Evolve to time t
t = np.pi / 2
psi_t = runge_kutta_4(psi0, H, t, dt=0.001)

print("Final state:", psi_t)
print("Probabilities:", np.abs(psi_t)**2)
```

4 Complete Example

Problem: Evolve a 3-qubit system under $H = X \otimes I \otimes I + I \otimes X \otimes I + I \otimes I \otimes X$ from initial state $|000\rangle$ to time $t = 1$.

```
import numpy as np
from hamiltonian_builder import hamiltonian_from_string
from runge_kutta_hamiltonian import runge_kutta_4

# Build Hamiltonian
H = hamiltonian_from_string("XII + IXI + IIX")

# Initial state |000>
psi0 = np.zeros(8, dtype=complex)
psi0[0] = 1.0

# Time evolution
t = 1.0
psi_final = runge_kutta_4(psi0, H, t, dt=0.001)

# Results
print("H shape:", H.shape)  # (8, 8) for 3 qubits
print("Initial state:", psi0)
print("Final state:", psi_final)
print("Norm:", np.linalg.norm(psi_final))  # Should be 1.0
```

5 Common Hamiltonian Examples

5.1 Transverse-Field Ising Model

$$H = -J \sum_{i=0}^{n-2} Z_i \otimes Z_{i+1} - h \sum_{i=0}^{n-1} X_i$$

```
n = 3
J, h = 1.0, 0.5
builder = HamiltonianBuilder()

# ZZ interactions
for i in range(n-1):
    ops = ['I'] * n
    ops[i], ops[i+1] = 'Z', 'Z'
    builder.add_term(ops, coefficient=-J)

# Transverse field
for i in range(n):
    ops = ['I'] * n
    ops[i] = 'X'
    builder.add_term(ops, coefficient=-h)

H = builder.build()
```

5.2 Heisenberg Model

$$H = \sum_{i=0}^{n-2} [J_x X_i \otimes X_{i+1} + J_y Y_i \otimes Y_{i+1} + J_z Z_i \otimes Z_{i+1}]$$

```
n = 3
Jx, Jy, Jz = 1.0, 1.0, 1.0
builder = HamiltonianBuilder()

for i in range(n-1):
    for pauli, J in [('X', Jx), ('Y', Jy), ('Z', Jz)]:
        ops = ['I'] * n
        ops[i], ops[i+1] = pauli, pauli
        builder.add_term(ops, coefficient=J)

H = builder.build()
```

6 Quick Reference Summary

Task	**Code**
Import modules	`from hamiltonian_builder import ...` `from runge_kutta_hamiltonian import ...`
Build H (string)	`H = hamiltonian_from_string("X@Z + Z@X")`

Build H (class)	`builder = HamiltonianBuilder()`
	`builder.add_term(['X', 'Z'], coefficient=1.0)`
	`H = builder.build()`

Time evolution	`psi_t = runge_kutta_4(psi0, H, t, dt=0.001)`
Check Hermitian	`np.allclose(H, H.conj().T)`
Check norm	`np.linalg.norm(psi)`

7 Important Notes

- All Hamiltonians must be Hermitian: $H = H^\dagger$
- State vectors are normalized: $\langle\psi|\psi\rangle = 1$
- For n qubits, Hamiltonian is $2^n \times 2^n$ matrix
- Smaller `dt` gives better accuracy but slower computation
- Energy is conserved during unitary evolution: $\langle\psi(t)|H|\psi(t)\rangle = \text{const}$