Simulating Strongly Driven Jaynes-Cummings Oscillator
Project Submission for PHY-354 Computational Physics

S Shri Hari, 3rd Year Undergraduate

ABSTRACT

The Jaynes-Cummings (JC) Hamiltonian is a theoretical model for a qubit interacting with a (quantized) EM Mode in an optical cavity. This model is applicable for a wide range of experiments involving both natural atoms and solid-state artificial atoms. In the presence of a drive, the system cannot be studied analytically unless the drive (perturbation) is small. As a result, numerical simulations are necessary for its study under strong drive. This project is aimed to simulate JC Hamiltonian under strong drive in a strongly dispersive, bad-cavity regime. The project is based of the paper by Lev S. Bishop, Eran Ginossar, and S. M. Girvin, titled "Response of the Strongly Driven Jaynes-Cummings Oscillator".

THEORY

Jaynes-Cummings Hamiltonian

The Jaynes-Cummings Hamiltonian is a model that describes a two-level system (qubit) interacting with the quantized EM mode in an optical cavity. It consists of the free Hamiltoninan of the qubit and the cavity (the EM Mode) and the Hamiltonian describing the interaction between the qubit and the field.

$$H_{qubit} = \frac{1}{2}\hbar\omega_q\sigma_z$$

Taking the zero-point energy to lie in the middle of the energy gap between the two states, the free Hamiltonian of the qubit is written accordingly

$$H_{field} = \hbar\omega_c a^\dagger a$$

The EM Field, on quantization, will have a Hamiltonian of the form similar to a Harmonic Oscillator with frequency ω_c, where the number operator $a^\dagger a$ represents the number of photons in the cavity.

$$H_{int} = g(a\sigma_+ + a^\dagger\sigma_-)$$

The Interaction Hamiltonian consists of two terms, $a\sigma_+$ and $a^\dagger\sigma_-$, where a and † act on the field and $\sigma_+ = |1\rangle\langle 0|$, $\sigma_- = |0\rangle\langle 1|$ act on the qubit. These terms are intuitive to understand on observing its action on the system. A state where the qubit is in ground state $|0\rangle$ will become excited due to σ_+, while simultaneously, the cavity will lose a photon (if present) due to the action of a. Similarly, a qubit in excited state, will lose photon by action of σ_- while simultaneously a new photon is added to the cavity by $a^\dagger$

With the presence of a drive, oscillating sinusoidally with frequency ω_d, the driven JC Hamiltonian has the form

$$H = \omega_c a^\dagger a + \frac{\omega_q}{2}\sigma_z + g(a\sigma_+ + a^\dagger\sigma_-) + \frac{\xi(t)}{\sqrt{2}}(a + a^\dagger) \tag{1}$$

Decoupling qubit and Environment in JC Hamiltonian

The terms in the interaction Hamiltonian can be decoupled with ease by noticing a feature of it; this Hamiltonian is a block diagonal matrix in the standard basis (Fock Basis). It is easy to see that every basis state $|n\rangle_{field} \otimes |i\rangle_{qubit}$ has only one state it can transition to and from. So, if we order the basis accordingly, we get a block diagonal form of the Hamiltonian where each block is a 2×2 matrix. On appropriate Unitary transformation to its eigenkets, the JC Hamiltonian has the form

$$\begin{aligned}\tilde{H} &= UHU^\dagger \\ &= \omega_c a^\dagger a + (\omega_c - \Delta)\frac{\sigma_z}{2} + \frac{\xi}{\sqrt{2}}(a + a^\dagger)\cos(\omega_d t)\end{aligned} \tag{2}$$

where $N = a^\dagger a + \sigma_z/2 + 1/2$, $\delta = \omega_q - \omega_c$, and $\Delta = (\delta^2 + 4g^2N)^{1/2}$ The Eigenstates of the Interaction Hamiltonian are called as dressed states.

Quantum Master Equation

In this study, we need to model a 'leaky' cavity, where photons leak out of the cavity at a specified rate κ. To model such probabilistic events, we need to use the following:

- Density Matrices that can capture classical mixtures of quantum systems.
- A Quantum Master Equation (Lindbladian) that describes the evolution of a mixed state (represented by Density Matrices) while accounting for probabilistic events (like the photon leaking out of the cavity).

Such processes usually translate to non-unitary evolution of the state. For this purpose, the Master Equation describing the evolution of the state $\rho(t)$ is

$$\dot{\rho(t)} = -i[\tilde{H(t)}, \rho(t)] + \kappa([a\rho, a^\dagger] + [a, \rho a^\dagger])/2 \tag{3}$$

Parameters of the System

The Driven JC Hamiltonian is studied under the following conditions:

1. Bad-Cavity Limit: Cavity Relaxation Rate κ larger than the qubit dephasing rates γ, γ_ϕ
2. Strong Dispersive Regime: A non-negligible shift of the cavity frequency (greater than cavity-linewidth $\chi = g^2/(\omega_d - \omega_c)$) due to the presence of qubit

3. Presence of a Strong Drive: A sinusoidal drive $\xi(t) = \xi \cos(\omega_d t)$ where $\xi \gg \xi_1 = \kappa/\sqrt{2}$

This gives us a 'hierarchy of scales' to make appropriate approximations/simplifications

$$\gamma, \gamma_\phi \ll \kappa \ll \chi \ll g \ll \delta \ll \omega_c \tag{4}$$

Due to the large cavity frequency, we will be dealing with time periods much smaller than the qubit decoherence times γ^{-1} and γ_ψ^{-1}. So, we can take the state of the qubit to be a constant of motion. For convenience, we choose the qubit to be fixed at ground state $\sigma_z = -1$ throughout the simulation time. This greatly reduces the complexity of the computation as we need not deal with product states.

Method of Quantum Trajectories

Simulating the evolution of density matrices is computationally intensive if the system is large or complex enough, due to the fact that the state is now represented as a square matrix and not a vector. As we only need information regarding the (average) values of observables/parameters across time, we can instead simulate the trajectories of multiple states starting from the same initial state. Note that they will evolve differently due to the probabilistic nature of the system (photon decay rate $\kappa > 0$). The Method of Quantum Trajectories is constructed to simulate the evolution of a realization of the system
For the given master equation,

$$\dot{\rho} = -i[H, \rho] + \sum_n \frac{1}{2}([C_n\rho, C_n^\dagger] + [C_n, \rho C_n^\dagger])$$

the evolution of such individual states can be described by the following non-Hermitian 'effective' Hamiltonian

$$H_{eff} = H - \frac{i}{2}\sum_n C_n^\dagger C_n \tag{5}$$

Due to the non-Hermitian term, we can conclude that the norm of a state $|\psi(t)\rangle$ must decay over time. If at time t, the ket has a unit norm, the norm at time $t + \delta t$ has a norm $1 - \delta p$ where

$$\delta p = \delta t \sum_n \langle\psi(t)| C_n^\dagger C_n |\psi(t)\rangle \ll 1 \tag{6}$$

upto a first order in time. The 'quantum jump' occurs with the probability δp where, the state transitions to a new state as described below

$$|\psi(t + \delta t)\rangle = \frac{C_i |\psi(t)\rangle}{\langle\psi(t)| C_i^\dagger C_i |\psi(t)\rangle} \tag{7}$$

with the index i chosen with a probability of

$$P_i(t) = \langle\psi(t)| C_i^\dagger C_i |\psi(t)\rangle / \delta p \tag{8}$$

TRANSLATING THEORY INTO CODE

Basis and Truncated Hilbert Space

The Hilbert Space here is infinite-dimensional and so are the wavefunctions and operators. As a result, we need to choose a small subset of the appropriate basis within which we can expect the state to be present. This is called a 'truncated' Hilbert Space
The Fock Basis is chosen for representing the states and operators in Matrix Form and the number of basis vectors are limited by $n \approx 1000$, starting from the ground state of cavity (zero photon).

Implementation of Quantum Trajectory Simulation

Simulating Quantum Trajectories upto first order in time is not feasible. So, the following algorithm is used to simulate trajectories of individual realizations of the system:

1. Choose a random number r between zero and one, representing the probability that a quantum jump occurs.
2. Integrate the Schrodinger Equation $i\frac{\partial}{\partial t}|\psi(t)\rangle = H_{eff}|\psi(t)\rangle$ until we reach a time τ where $\langle\psi(\tau)|\psi(\tau)\rangle = r$
3. After $t = \tau$, the state undergoes a quantum jump described previously. The index i is chosen such that i is the smallest number satisfying

$$\sum_{j=1}^{i} P_j(\tau) \geq r$$

4. The new state obtained will be now renormalized, another new random number r is drawn, and the process is repeated all over again till we reach the final simulation time.

Time Dependent Hamiltonian

Hamiltonians are separated into time-independent and time-dependent parts, where the dependence in time is restricted to only coefficients of operators

$$H = H_0 + \sum_{i=1} c_i(t) \cdot H_i \tag{9}$$

Further Optimizations

Running Multiple Trajectories Simultaneously

Instead of simulating trajectories one after the other, all the states undergo the time evolution simultaneously followed by quantum jump (if conditions are satisfied).

Computing Time Evolution of the state

The time evolution of the state between time τ and time $\tau+\delta t$ is according to the Schrodinger Equation

$$i\frac{\partial}{\partial t}\left|\psi(t)\right\rangle = H\left|\psi(t)\right\rangle \tag{10}$$

The equations are integrated numerically in two methods:

1. Using RK4 routine: Viable for Time Independent Hamiltonian
2. Using Spectral Decomposition: For Time Dependent Hamiltonians

The problem with time-dependent Hamiltonian is that they need to be generated at every point in time, which takes a lot of resources (memory or time) for large dimensional spaces. Also, the frequency of oscillation of states increases as the maximum energy of the state increases (governed by the domain of the Truncated Hilbert Space)
There are a couple of observations to make specific about this system:

1. The time-Dependent Hamiltonian is periodic, and so we can reduce the function calls to generate the Hamiltonians by generating them beforehand for fixed point in a time period of it's oscillation and call accordingly.
2. We can incorporate the oscillations in the system by changing the basis to the eigenbasis of the Hamiltonian (which we will consider to be constant during the time δt) and computing it's evolution (which is simply a multiplication by a diagonal matrix)

The above points motivates using the method of spectral decomposition

Spectral Decomposition of Hamiltonian

For a time-independent Hamiltonian, the eigenstates of the Hamiltonian have a simple time evolution.

$$i\frac{\partial}{\partial t}\left|\lambda\right\rangle = H\left|\lambda\right\rangle \tag{11}$$

$$H\left|\lambda\right\rangle = E_\lambda\left|\lambda\right\rangle \rightarrow \left|\lambda(t)\right\rangle = e^{-iE_\lambda t}\left|\lambda(0\right\rangle \tag{12}$$

This makes computing time evolution of the state easier without worrying about the frequency of oscillations. So, we can compute the matrix that takes a state $\left|\psi(t)\right\rangle$ to $\left|\psi(t+\delta t)\right\rangle$ by the following:

$$H = Q\Lambda Q^{-1} \tag{13}$$

- Q - $n\times n$ matrix whose ith column corresponds to the ith Eigenvector of H. Invertible as Eigenvectors are linearly independent (unless zero Eigenvalue exists)
- Λ - $n\times n$ Diagonal Matrix whose diagonal elements are corresponding Eigenvectors

$$\left|\psi(t)\right\rangle = Qe^{-it\Lambda}Q^{-1}\left|\psi(t+\delta t)\right\rangle \tag{14}$$

The required matrices can be computed beforehand, and so can drastically reduce the computation time.

CODE DOCUMENTATION

- All the functions and routines are stored in the `qu-eig.py`.
- The only required dependencies are NumPy. Sometimes, the QuTiP (Quantum Toolbox in Python) package is used as reference/comparison
- Matplotlib's PyPlot is used to visualize data.

Generating States

- `fock(n, k)`:
  Generates a 2-D $n \times 1$ NumPy matrix with kth element one and the remaining zero. Represents $|()\, k)$ in Fock Basis (in a Truncated Hilbert Space of n dimensions)

Generating Operators

All the functions are meant to generate 2-D $n \times n$ NumPy matrices, representing operators using Fock Basis (in a Truncated Hilbert Space of n dimensions)

- `identity(n)`: Identity Matrix **I**
- `zeros(n)`: Zero Matrix
- `destroy(n)`: The annihilation operator a, where $a\,|n\rangle = \sqrt{n}\,|n-1\rangle$
- `\number(n)`: The number operator $N = a^{\dagger}a$, where $N\,|n\rangle = n\,|n\rangle$

Transforming Operators

- `dag(oper)`:
  Input: 2-D NumPy Matrix `oper`
  Output: 2-D NumPy Matrix; the conjugate transpose of `oper`

Expectation Values

- `expect(oper, psi)`
  Input: 2-D $n \times n$ NumPy Matrix `oper`; 2-D $n \times 1$ NumPy Matrix `psi`
  Output: Complex Scalar; The expectation value $\langle\psi|\,O\,|\psi\rangle$
- `expectAvg(oper, psi)`
  Input: 2-D $n \times n$ NumPy Matrix `oper`; 2-D $n \times m$ NumPy Matrix `psi`, each row represents a single state
  Output: Complex Scalar; The expectation value $\langle\psi|\,O\,|\psi\rangle$ averaged across all input states

RK4 Solver `rk4Solve|`

Note: `rk4Step` is a subroutine for `rk4Solve`

Input

- `H`: The Hamiltonian of the System. Must be of the form `[H0, [H1, H1_coeff]]` where
  - `H0`: 2-D $n \times n$ NumPy Matrix; Time Independent Term
  - `H1`: 2-D $n \times n$ NumPy Matrix; Time Dependent Term (the coefficient is dependent in time not the operator itself)
  - `H1_coeff(t, H_args)`: Function that returns a complex scalar that represents the time-dependent coefficient

$$H = H_0 + H_{1coeff}(t)H_1$$

- `psi0`: 2-D $n \times 1$ NumPy Matrix; The starting state of the system(s)
- `cyclePeriod`: Positive Real floating scalar; The time period of the periodic oscillation of `H1_coeff`
- `cycleRes`: Positive Integer; The number of points in a single time period of oscillation to be taken
- `cycleCount`: Positive Integer; The number of cycles to simulate
- `c_op`: 2-D $n \times n$ NumPy Matrix; Collapse Operator in Effective Hamiltonian
- `e_op`: 2-D $n \times n$ NumPy Matrix; Expectation Values of states to compute (apart from Number operator)
- `H_args`: Python Dictionary; Additional Arguments to be passed to `H1_coeff` to generate coefficients
- `ntraj`: Positive Integer; The number of trajectories(realizations of systems) to simulate
- `showProgress`: Boolean; Option to print text to show progress in simulation

Output

A 2-D $n \times 3$ NumPy matrix where the first column contains the time, the second and third column contains the expectation values of number operator and `e_op` respectively.

Spectral Decomposition Solvers `eigSolve|`

Note: `genStepOper` is a subroutine for `eigSolve` The input and output is same as in `rk4Solve`

Input

- `H`: The Hamiltonian of the System. Must be of the form `[H0, [H1, H1_coeff]]` where
  - `H0`: 2-D $n \times n$ NumPy Matrix; Time Independent Term
  - `H1`: 2-D $n \times n$ NumPy Matrix; Time Dependent Term (the coefficient is dependent in time not the operator itself)
  - `H1_coeff(t, H_args)`: Function that returns a complex scalar that represents the time-dependent coefficient

$$H = H_0 + H_{1coeff}(t)H_1$$

- `psi0`: 2-D $n \times 1$ NumPy Matrix; The starting state of the system(s)
- `cyclePeriod`: Positive Real floating scalar; The time period of the periodic oscillation of `H1_coeff`
- `cycleRes`: Positive Integer; The number of points in a single time period of oscillation to be taken
- `cycleCount`: Positive Integer; The number of cycles to simulate
- `c_op`: 2-D $n \times n$ NumPy Matrix; Collapse Operator in Effective Hamiltonian
- `e_op`: 2-D $n \times n$ NumPy Matrix; Expectation Values of states to compute (apart from Number operator)
- `H_args`: Python Dictionary; Additional Arguments to be passed to `H1_coeff` to generate coefficients
- `ntraj`: Positive Integer; The number of trajectories(realizations of systems) to simulate
- `showProgress`: Boolean; Option to print text to show progress in simulation

Output

A 2-D $n \times 3$ NumPy matrix where the first column contains the time, the second and third column contains the expectation values of number operator and `e_op` respectively.