30 January 2026

QFlux: An Open-Source Toolkit for Quantum Dynamics Simulations on Quantum Computers. Part V - Adaptive Variational Quantum Algorithms for Open Quantum Systems

Saurabh Shivpuje[1], Alexander V Soudackov[2], Xiaohan Dan[2], Yuchen Wang[1], Brandon C Allen[2], Delmar G A Cabral[2], Zixuan Hu[1], Ningyi Lyu[2], Eitan Geva[3], Victor S Batista[2,4], Sabre Kais[5]

1. Department of Chemistry Purdue University
2. Department of Chemistry Yale University
3. Department of Chemistry University of Michigan
4. Yale Quantum Institute Yale University
5. Department of Electrical and Computer Engineering North Carolina State University

Abstract

Simulating open quantum systems remains one of the most demanding problems in quantum dynamics, as environmental interactions lead to non-unitary evolution that challenges standard quantum simulation techniques. Variational quantum algorithms (VQAs) offer a practical way forward by combining classical optimization with quantum hardware, making them well suited for near-term devices. Here, in Part V of the QFlux implementation series, we introduce adaptive variational-ansatz methods for solving the Lindblad master equation. This tutorial presents the Python-based qmad module of the QFlux toolkit, which is designed to help students and researchers easily build and explore adaptive VQA strategies. The tutorial walks through each step of the workflow and applies the methods to two illustrative examples. The first is the amplitude-damping channel, which models spontaneous emission in a two-level system. The second is the Fenna-Matthews-Olson (FMO) complex, a prominent model for excitonic energy transfer in photosynthesis. The two workflows utilizing adaptive variational approaches implemented here are the unrestricted adaptive variational quantum dynamics (UAVQD) scheme and the stochastic Schrödinger equation (SSE) based

trajectory method. By working through these examples, readers gain both intuition and hands-on experience with variational techniques for simulating dissipative quantum dynamics.

QFlux: An Open-Source Toolkit for Quantum Dynamics Simulations on Quantum Computers. Part V - Adaptive Variational Quantum Algorithms for Open Quantum Systems

Saurabh Shivpuje,[†] Alexander V. Soudackov,[‡] Xiaohan Dan,[‡] Yuchen Wang,[†] Brandon C. Allen,[‡] Delmar G. A. Cabral,[‡] Zixuan Hu,[†] Ningyi Lyu,[‡] Eitan Geva,[¶] Victor S. Batista,[*,‡,§] and Sabre Kais[*,‖]

[†]*Department of Chemistry, Purdue University, West Lafayette, Indiana 47907, USA*

[‡]*Department of Chemistry, Yale University, New Haven, CT 06520, USA*

[¶]*Department of Chemistry, University of Michigan, Ann Arbor, MI 48109, USA*

[§]*Yale Quantum Institute, Yale University, New Haven, CT 06511, USA*

[‖]*Department of Electrical and Computer Engineering, Department of Chemistry , North Carolina State University, Raleigh, North Carolina 27606, USA*

E-mail: victor.batista@yale.edu; skais@ncsu.edu

Abstract

Simulating open quantum systems remains one of the most demanding problems in quantum dynamics, as environmental interactions lead to non-unitary evolution that challenges standard quantum simulation techniques. Variational quantum algorithms (VQAs) offer a practical way forward by combining classical optimization with quantum hardware, making them well suited for near-term devices. Here, in Part V of the

QFlux implementation series, we introduce adaptive variational-ansatz methods for solving the Lindblad master equation. This tutorial presents the Python-based `qmad` module of the **QFlux** toolkit, which is designed to help students and researchers easily build and explore adaptive VQA strategies. The tutorial walks through each step of the workflow and applies the methods to two illustrative examples. The first is the amplitude-damping channel, which models spontaneous emission in a two-level system. The second is the Fenna–Matthews–Olson (FMO) complex, a prominent model for excitonic energy transfer in photosynthesis. The two workflows utilizing adaptive variational approaches implemented here are the unrestricted adaptive variational quantum dynamics (UAVQD) scheme and the stochastic Schrödinger equation (SSE) based trajectory method. By working through these examples, readers gain both intuition and hands-on experience with variational techniques for simulating dissipative quantum dynamics.

1 Introduction

Open quantum systems introduce an additional layer of complexity beyond that encountered in isolated, closed systems, primarily due to interactions between a system and its surrounding environment. Unlike closed quantum systems, which evolve only through their internal Hamiltonian, open quantum systems interact with their surroundings. These external influences cause dissipation, decoherence, and time evolution that is no longer unitary. These effects complicate the design of efficient quantum algorithms,[1–6] yet they are unavoidable in realistic physical settings. Indeed, many systems of interest in chemistry and physics operate inherently as open systems,[7–10] making the development of practical simulation strategies for open-system dynamics an essential goal of quantum computing research.

Quantum computing has emerged as a powerful computational paradigm with potential applications across a wide range of disciplines, including the simulation of open quantum systems.[1–4,11–16] Earlier installments of the **QFlux**[17] tutorial series introduced the Lindblad

master equation and its simulation using dilation-based approaches.[1] In this part, we shift focus to *variational quantum algorithms* for open quantum systems by formulating transformed representations of the Lindblad master equation that are amenable to variational treatment on quantum hardware.

A central challenge in simulating open quantum dynamics is its fundamentally non-unitary nature. This presents a significant obstacle, since the elementary operations available on quantum hardware are inherently unitary.[1,2,12,18] To address this mismatch, a growing family of hybrid quantum-classical methods known as *variational quantum algorithms* (VQAs) has been developed.[4,19,20] These approaches draw inspiration from classical variational techniques long used to approximate the dynamics of complex many-body systems.[21,22] In the context of noisy intermediate-scale quantum (NISQ) devices, VQAs are particularly attractive because they rely on shallow circuits and parametrized ansätze optimized through classical feedback loops.[21,23] While VQAs were initially developed for energy minimization and unitary dynamics, they have recently been extended to simulate dissipative and open-system processes.[4,20]

Early implementations of the Variational Quantum Eigensolver (VQE) typically relied on fixed-form ansätze, such as the unitary coupled-cluster construction. Although successful for small systems, these approaches often suffer from limited accuracy and unfavorable scaling, requiring a polynomial increase in both circuit depth and the number of variational parameters as system size grows.[24,25] To overcome these limitations, *adaptive* VQE methods have been introduced, in which the ansatz is constructed dynamically in response to the problem at hand.[5] By tailoring the ansatz to the evolving system state, adaptive approaches can achieve higher accuracy with significantly shallower circuits.

In this work, we will discuss two distinct approaches which utilize adaptive ansatz: the first is an unrestricted, vectorized *adaptive variational quantum dynamics* (AVQD) scheme, which iteratively appends operators from a predefined pool to ensure that the McLachlan distance remains below a prescribed threshold during time evolution. The second approach

is based on a *stochastic Schrödinger equation* (SSE) formulation, which avoids vectorization and preserves the original system qubit count. This trajectory-based method is therefore more suitable for NISQ devices, while reproducing Lindblad dynamics through ensembles of pure-state evolutions.[20]

Although adaptive ansatz frameworks have been proposed in several recent studies, their practical implementation often remains challenging. Many works provide detailed theoretical descriptions or pseudo-algorithms, but the absence of readily executable code poses a barrier for researchers seeking to apply these methods in practice.[20,26,27] While robust quantum optics and open-system packages exist in languages such as Julia,[28] much of the quantum computing community relies on Python-based software development kits.[29–32]

To address this gap, we introduce the `qmad` module within the **QFlux** framework (available via `pip install qflux`), a Python-based[33] toolkit designed to support adaptive variational quantum algorithms for open-system dynamics. `qmad` module provides a unified and accessible implementation of the methods developed in this work, enabling users to incorporate adaptive variational techniques directly into their simulation workflows. The module exposes the core algorithmic components discussed throughout this paper and allows users to explore different adaptive ansatz constructions across a broad range of quantum systems.

To illustrate these methods, we consider two representative examples. The first is the amplitude-damping channel, which models spontaneous emission in a two-level system.[1,9] The second example is the Fenna–Matthews–Olson (FMO) complex, a chemically realistic system widely studied in the context of light-harvesting and energy transfer in photosynthetic bacteria.[9,10] For both cases, Lindblad dynamics are first simulated using the QuTiP framework to establish classical reference results.[32] We then apply the unrestricted adaptive variational quantum dynamics (UAVQD) approach to the amplitude-damping model,[9,20] followed by the SSE-based variational trajectory method for the FMO complex.[4,9] Together, these examples demonstrate how adaptive variational algorithms can be used to simulate realistic open-system dynamics within the QFlux framework.

Finally, while adaptive variational frameworks can, in principle, be extended to treat non-Markovian dynamics, the methods implemented in this installment focus on Lindblad-form generators and therefore describe Markovian open-system evolution.

In the following section, we first establish classical reference simulations of Lindblad dynamics, which serve as benchmarks for the adaptive variational quantum algorithms introduced thereafter.

2 Numerical Simulations on Classical Computers

Before delving into the key quantum algorithms in this work, it is crucial to first become acquainted with the numerical simulations that can be performed on classical computers. It is a best practice to run simpler trial examples using numerical simulations before testing quantum algorithms on quantum computers. The results obtained from such simulations, often referred to as "exact results", serve as a benchmark for evaluating the accuracy of quantum algorithms and quantum computers.

In this section, we demonstrate the simulation of the Lindblad master equation[32,34]

$$\frac{\partial \rho(t)}{\partial t} = -\frac{\mathrm{i}}{\hbar}[H, \rho(t)] + \sum_n \frac{1}{2}\gamma_n [2L_n \rho(t) L_n^\dagger - \rho(t) L_n^\dagger L_n - L_n^\dagger L_n \rho(t)]. \tag{2.1}$$

This equation describes the time-evolution of the density matrix $\rho(t)$ for an open quantum system, accounting for both unitary evolution and dissipative effects. The first term, $-\frac{\mathrm{i}}{\hbar}[H, \rho(t)]$, represents the system's Hamiltonian evolution, while the second term includes Lindblad operators L_n with rate coefficients γ_n, modeling environmental interactions like dissipation or decoherence. Together, they form the Lindblad master equation, a fundamental tool in quantum dynamics for describing systems that interact with external environments. To illustrate this equation and introduce the examples presented in this work, we simulate the dynamics of an amplitude-damping channel and then proceed to simulate the dynamics of an FMO complex system.

2.1 for the Lindblad master equation

An effective way to obtain numerically exact solutions of the Lindblad master equation is to use a dedicated , such as QuTiP.[32,34] QuTiP provides a built-in Lindblad solver, `mesolve`, which relies on SciPy's complex-valued VODE ("zvode") integrator.[35] The integrator offers two main methods: Adams, a variable-step predictor–corrector scheme suited for non-stiff problems, and BDF, a backward differentiation formula appropriate for stiff systems. Users need only specify the system parameters and desired outputs; the software handles the numerical integration and returns the computed dynamics.

When using `mesolve` as shown in Script S.1.1, key components must be provided: the system Hamiltonian H, the initial density matrix $\rho(0)$, a list of times for dynamic simulation, and *collapse operators* `c_ops`, defined as $\sqrt{\gamma_n}L_n$. If no collapse operators are given, the solver propagates the Liouville equation of the pure system. Additionally, users must specify the output instructions, particularly the operators `e_ops` whose expectation values are to be calculated. With these quantities defined, `mesolve` generates time-dependent expectation values for the given operators by propagating either the Liouville equation or the Lindblad master equation.

2.2 Amplitude Damping Channel

The amplitude-damping channel illustrated here models spontaneous emission and energy dissipation, which are among the dominant noise processes affecting quantum systems. This process is ideally suited for benchmarking numerica solutions since it can be solved analytically as shown in Section S.2.1 of the Supporting Information.

The amplitude damping dynamics is governed by a decay rate γ, which determines how rapidly the system transitions from the *excited state* $|1\rangle$ to the *ground state* $|0\rangle$. Throughout this work, we adopt the computational basis ordering commonly used in quantum optics. Accordingly, in Script S.2.1, the lowering operator is defined as $\sigma^- = |0\rangle\langle 1|$ and implemented as `sm`, which induces the "downward" transition from $|1\rangle$ to $|0\rangle$ and models the decay of

population from the excited to the ground state.

The initial state, defined as a superposition of $|0\rangle$ and $|1\rangle$ and normalized for proper behavior, is converted to a density matrix `rho0` to allow for compatibility with the master equation solver. To capture the dynamics of energy dissipation, a collapse operator `c_ops` is constructed by scaling the lowering operator with $\sqrt{\gamma}$, thus modeling the amplitude damping at rate γ.

For the time-evolution, the code utilizes QuTiP's `mesolve` function to simulate the system's behavior over a specified time array, 'times', with each time step set at $dt = 0.1$ ps, culminating at a final time `tf = 1000` ps. The solver computes the evolution of `rho0` under the influence of `c_ops`, providing a realistic model of energy dissipation. To observe the population dynamics, two projectors, `proj_ground` and `proj_excited`, are defined to extract the populations of the ground and excited states at each time step. These populations are then plotted to reveal the behavior of the system as it undergoes amplitude damping, offering valuable insights into the effects of spontaneous emission on a simple two-level quantum system, an essential foundation for understanding quantum noise and the challenges of mitigating decoherence in quantum computing applications.

Figure 1 depicts the temporal evolution of state populations in a two-level quantum system undergoing amplitude damping. At the initial time, the system exhibits a dominant population in the excited state, indicating a higher occupancy of $|1\rangle$. As time advances, a continuous relaxation process is observed, characterized by the progressive transfer of population from the excited state to the ground state.

This behavior reflects the amplitude damping process, where the population in the excited state $|1\rangle$ dissipates over time due to energy loss (spontaneous emission), resulting in an increase in the ground state $|0\rangle$ population. By the end of the simulation (at around 1000 ps), the ground state population approaches a higher value, while the excited state population diminishes, demonstrating the effect of continuous decay. This trend aligns with the characteristics of amplitude damping, as the system loses energy and stabilizes predominantly in

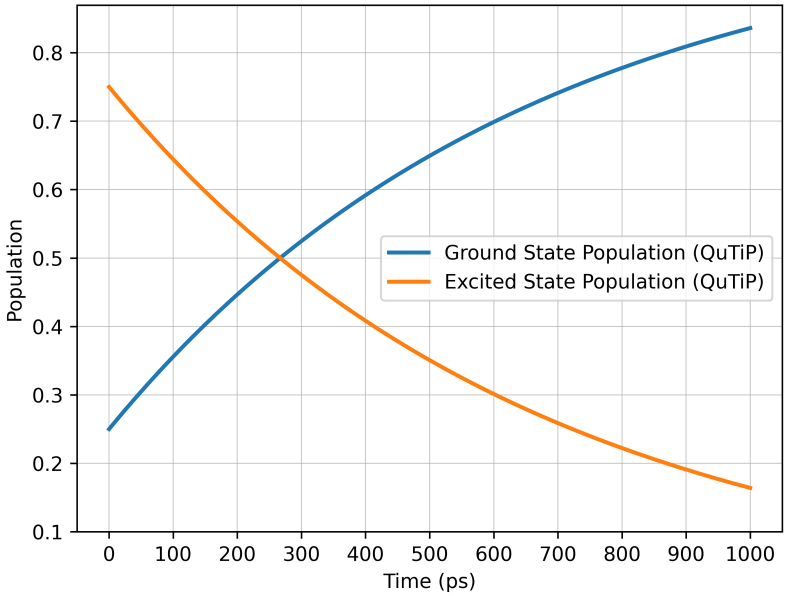

Figure 1: Population dynamics of a two-level quantum system under amplitude damping. The blue curve illustrates the population of the ground state $|0\rangle$, while the orange curve represents the excited state $|1\rangle$. Initially excited, the system relaxes to the ground state via damping.

its ground state, highlighting the impact of noise on quantum states over time.

2.3 FMO Complex

Another example of using numerical calculations to study a real-world system[36] is the Fenna-Matthews-Olson (FMO) complex, a well-studied pigment-protein complex (PPC), prominent in quantum computations. Found in green sulfur bacteria, the FMO complex facilitates excitonic energy transfer (EET) between chromophores.[37] Within its intricate protein scaffold structure, chromophores are optimally packed, enabling highly efficient energy transfer between excitations.[38,39]

In every FMO complex entity, there are seven chromophore sites. The reaction center is located near site 3, while photoexcitation typically occurs at either site 1 or site 6.[15,40,41] The excited state energy from site 1 or 6 is then transferred to the reaction center through interactions with neighboring sites. This efficient energy transfer is facilitated by the close proximity and optimal arrangement of the chromophores within the complex.[38]

To study this quantum mechanical process, we will consider only one of the multiple pathways of excitation, i.e. excitation starting from site 1 and traveling to the reaction center via site 3. This brings the total number of possible states to 5: 3 chromophore states in the pathway, including ground and sink states. Hamiltonian for this considered pathway is:

$$H = \sum_{i=0}^{4} \varepsilon_i \sigma_i^+ \sigma_i^- + \sum_{i \neq j} J_{ij} (\sigma_i^+ \sigma_j^- + \sigma_j^+ \sigma_i^-). \tag{2.2}$$

In the above Hamiltonian equation, σ_i^+ and σ_i^- are Pauli raising and lowering operators, respectively, for the state i. ε_i denotes their creation and annihilation energy, and J_{ij} denote the coupling strength between the two states i, j. For calculations here, we utilize the Hamiltonian matrix as provided by Hu et al.[10] and mentioned in the accompanying code (Script S.3.1).

The Lindblad master equation to simulate the FMO complex excitation pathway is given as:

$$\frac{\partial \rho_{\text{tot}}(t)}{\partial t} = -\mathrm{i}[H, \rho_{\text{tot}}(t)] + \sum_n (L_n \rho_{\text{tot}}(t) L_n^\dagger - \frac{1}{2} L_n^\dagger L_n \rho_{\text{tot}}(t) - \frac{1}{2} \rho_{\text{tot}}(t) L_n^\dagger L_n). \tag{2.3}$$

where ρ_{tot} refers to the density matrix encompassing the populations and correlations between the system (ground state and excited states) and the reaction center (sink state). The operators L_n are the jump operators, which account for the 7 dissipation channels present in the pathway. One must note that the rate constants for the 7 different channels are wrapped within L_n as $\sqrt{\text{rate constant}} \times L_n \rightarrow L_n$. These channels include the dephasing rate parameter, denoted as α, applied to sites 1 through 3. The dissipation rate parameter, β, also pertains to sites 1 through 3 and describes transitions from these chromophores to the ground state. Additionally, γ represents the dissipation rate from site 3 to the sink state 4. These parameters are crucial for modeling jump operators as shown in the code (Script S.3.1) and understanding the energy transfer dynamics within the FMO complex.

Now, each site's population is studied using the diagonal elements of the density matrix,

which are obtained through projection measurements in the computational subspace. This procedure is detailed in Script S.3.2, which also includes the code to plot the results. In Fig. 2, the population change over time among all five states is illustrated. Observing how the populations transfer among the states as time progresses provides valuable insights.

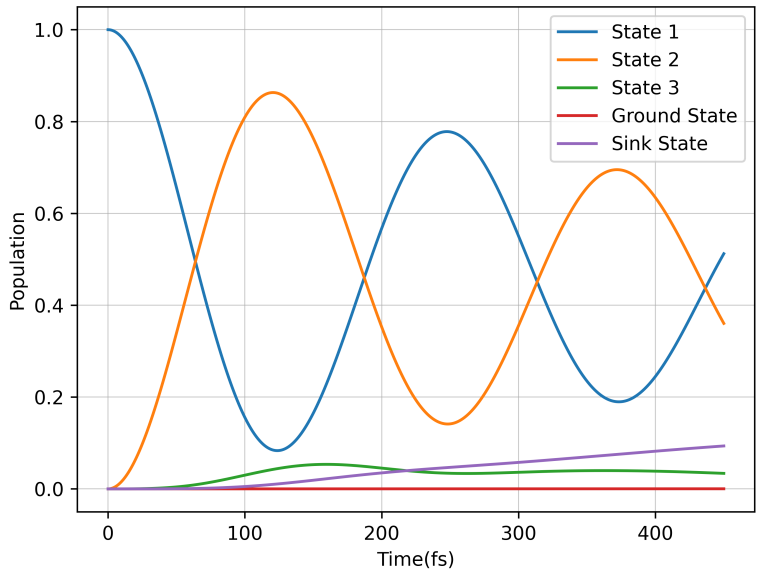

Figure 2: Time-evolution of populations for chromophore sites in Fenna-Matthews-Olson (FMO) complex during energy transfer.

Figure 2 shows that following initialization at site 1 of the FMO complex, the excitation is coherently redistributed among the intermediate chromophore sites before being funneled toward site 3 and subsequently transferred to the sink state, which represents the reaction center. The gradual depletion of population from the excited sites, together with the accumulation in the sink state, reflects the combined effects of coherent coupling, dephasing, and irreversible dissipation encoded in the Lindblad operators. This behavior is consistent with the role of the FMO complex as an efficient energy-transfer conduit in photosynthetic systems.

Having established reliable classical benchmarks for the Lindblad dynamics using numerical simulations, we now turn to variational quantum approaches for simulating open

quantum systems. In the following section, we introduce adaptive variational quantum dynamics methods that leverage parametrized quantum circuits to approximate non-unitary time evolution, using the amplitude-damping channel as a representative example.

3 Variational Simulations of Lindblad Dynamics

Adaptive Variational Quantum Dynamics Simulations (AVQDS), which fall under the category of variational simulations, offer a method for leveraging an ongoing improvements of quantum hardware that offer scalability and reduced noise to address systems governed by open quantum dynamics.[20] Here, we focus on the unrestricted and vectorized variation of AVQDS, which is particularly useful for simulating systems that can evolve in many possible pathways in a non-unitary manner. In this work, we will limit our discussion to introducing this method for a relatively simple example. We begin by discussing the fundamental principles of unrestricted-vectorization variant of AVQDS and its uniqueness.[9]

Next, we provide an example involving an amplitude damping channel. This example showcases the development of an expressible ansatz, which is a critical component in the AVQDS framework. The ansatz unitary serves as a trial solution, which is iteratively improved to approximate the desired quantum state evolution. Additionally, we delve into the creation of the associated operator pool, which consists of various quantum operations (gates) that can be applied to the system. This operator pool is essential for the adaptive nature of the algorithm, allowing it to dynamically adjust and refine the ansatz unitary based on the system's specific needs.

3.1 Vectorized Effective Hamiltonian

The initial step in developing a computational framework for this method involves converting the conventional Lindblad Master equation (Eq. (2.1)) into an effective Schrodinger equation. The reformulated effective Hamiltonian based evolution equation in AVQDS method

is presented as follows:

$$\frac{\partial\,|\nu_\rho(t)\rangle}{\partial t} = -\mathrm{i}H_{\text{eff}}(t)|\nu_\rho(t)\rangle, \tag{3.1}$$

where,

$$\rho \to |\nu_\rho\rangle = [\rho_{11}, \ldots, \rho_{1N}, \rho_{21}, \ldots, \rho_{2N}, \ldots, \rho_{N1}, \ldots, \rho_{NN}]^T, \tag{3.2}$$

$$H_{\text{eff}} = [\mathbb{I} \otimes H - H^T \otimes \mathbb{I}] + \mathrm{i}\sum_n \left[L_n^* \otimes L_n - \frac{1}{2}(\mathbb{I} \otimes L_n^\dagger L_n + L_n^T L_n^* \otimes \mathbb{I})\right]. \tag{3.3}$$

Here, the Lindblad operator L_n embodies γ_n as $\sqrt{\gamma_n}L_n \to L_n$.

The effective Hamiltonian is further split based on hermicity, into a Hermitian and an anti-Hermitian components as described next:

$$H_{\text{eff}} = H_e - \mathrm{i}H_a \tag{3.4}$$

The components after the decomposition are defined as follows:

$$H_e = \mathbb{I} \otimes H - H^T \otimes \mathbb{I}, \tag{3.5}$$

and

$$H_a = -\mathrm{i}\sum_n \left[L_n^* \otimes L_n - \frac{1}{2}(\mathbb{I} \otimes L_n^\dagger L_n + L_n^T L_n^* \otimes \mathbb{I})\right]. \tag{3.6}$$

The python functions responsible for this vectorization process and the decomposition of the effective Hamiltonian are provided in Script S.4.1.

3.2 Unrestricted adaptive procedure and system evolution

Following the vectorization of the density matrix and the determination of an effective Hamiltonian, we arrive at the core of this method: developing a quantum circuit (ansatz) that accurately describes the evolving quantum state of the system. To achieve a close approximation, the quantum circuit is parameterized for each timestep according to the following relation:

$$|\nu_\rho(t)\rangle \approx |\phi(t)\rangle = \prod_{l=1}^{k} \mathrm{e}^{-\mathrm{i}\theta_l(t)O_l} |\psi_i\rangle \,, \tag{3.7}$$

where $|\phi(t)\rangle$ is an approximated state obtained by applying unitary gates $\mathrm{e}^{-\mathrm{i}\theta_l(t)O_l}$ to the initial state $|\psi_i\rangle$. The operators O_l represent the ansatz operators added adaptively to the l^{th} layer of the circuit (discussed in detail in the next subsection) and $\theta_l(t)$ are real tunable parameters. The evolution of the state is achieved by tuning the parameters $\theta_l(t)$ such that the distance between the ideal evolution and the evolution induced by $\theta_l(t)$ is minimized. This is suggested by the McLachlan's variational principle[42] expressed as

$$\delta \left\| \frac{\partial \, |\phi(\boldsymbol{\theta}(t))\rangle}{\partial t} + \mathrm{i} H_{\text{eff}} |\phi(\boldsymbol{\theta}(t))\rangle \right\|^2 = 0. \tag{3.8}$$

The solution to the above equation is given by:

$$\mathbf{M}(t)\dot{\boldsymbol{\theta}}(t) = \mathbf{V}(t), \tag{3.9}$$

where the elements of the matrix $\mathbf{M}$ and components of the vector $\mathbf{V}$ are given by the following expressions:

$$M_{kj}(t) = 2\,\mathrm{Re}\left[\frac{\partial \langle \phi(\theta(t))|}{\partial \theta_k(t)} \frac{\partial |\phi(\theta(t))\rangle}{\partial \theta_j(t)} + \langle \phi(t)| \frac{\partial |\phi(\theta(t))\rangle}{\partial \theta_k(t)} \langle \phi(t)| \frac{\partial |\phi(\theta(t))\rangle}{\partial \theta_j(t)} \right] \tag{3.10}$$

$$V_k(t) = 2\,\mathrm{Im}\left[\langle H_{\text{eff}} \rangle \langle \phi(\theta(t))| \frac{\partial |\phi(\theta(t))\rangle}{\partial \theta_k(t)} + \frac{\partial \langle \phi(\theta(t))|}{\partial \theta_k(t)} H_{\text{eff}} |\phi(t)\rangle \right] \tag{3.11}$$

The calculations are performed at every timestep to determine the evolution trajectory of the system over a given interval of time. In addition to the parameter tuning in the adaptive procedure, the ansatz is updated by adding an operator from an operator pool to maintain the McLachlan distance below a specified limit. However, this lower bound distance is not always known a priori. This is where the unrestricted flavor of AVQDS was introduced.[20] Instead of fixing the threshold to a specific value, an operator is appended to the circuit

at each timestep such that it lowers the McLachlan distance below a relative threshold. This approach ensures that the distance attains the lowest possible value of the McLachlan distance. All these calculations are implemented through various Python functions. It is recommended to initially overlook these functions and run the codes as described in the following section, examining these functions only if modifications are needed.

3.3 Defining a Pool of Operators

As previously discussed, it is crucial for the algorithm to have a specified pool of operators from which it can heuristically select the appropriate candidate for the ansatz. The choice of the operator pool is specific to the system of interest. For a simple example such as the amplitude damping benchmark, a basic pool of one- and two-qubit Pauli/identity operators is chosen as these operators form a complete basis for the vectorized density matrix of the two-level system.

The actual implementation involves defining a function to build the operator pool and another function to create the ansatz with these operators. Script S.5.1 provides the code for defining a pool of operators. The function `build_pool` generates a pool of Pauli operators by iterating over combinations of qubit indices and Pauli operators for systems that involve multiple qubits. The ansatz function then uses this pool to construct the ansatz for the given initial state. This setup allows the algorithm to dynamically select and apply the most suitable operators from the pool to accurately model the evolving quantum state.

3.4 Amplitude Damping Channel

To illustrate the AVQDS scheme, we present a simulation of the two-level amplitude damping channel as a benchmark implementation. For this system, we need to predefine a pool of operators to be used in building an ansatz. This pool includes both single-qubit and two-qubit Pauli operators. Specifically, the single-qubit Pauli operators involve rotations around the X, Y, and Z axes, parameterized by the angle $\theta(t)$. To capture entanglement and

correlation, for example with an ancilla qubit, the two-qubit operators are introduced from the pool of combinations given by $P_{\text{two}} = \mathrm{e}^{-\frac{i\theta P_i \otimes P_j}{2}}$, where $P_i, P_j \in \{X_i, Y_i, Z_i\}$. This selection allows for dynamic evolution and accurately captures the entanglement and correlations in the system. The relevant code, showcased in Scripts S.5.1 and S.6.1, demonstrates how to provide inputs for the system to execute the code. It is important to note that these are just snippets of the complete code. The functions mentioned in the following script, such as `Ansatz` and `solve_avq`, which perform the ansatz construction and simulation process respectively, consist of relatively long lines of code with many supporting functions. For a detailed view, one can refer to the **QFlux** documentation.[17]

Figure 3 compares the approximate population dynamics obtained from the unrestricted adaptive variational quantum dynamics (UAVQD) method with the exact Lindblad solution computed using QuTiP which agrees with the analytical solution. The close agreement between the two results demonstrates that the adaptive variational ansatz accurately captures the dissipative dynamics of the amplitude-damping channel.

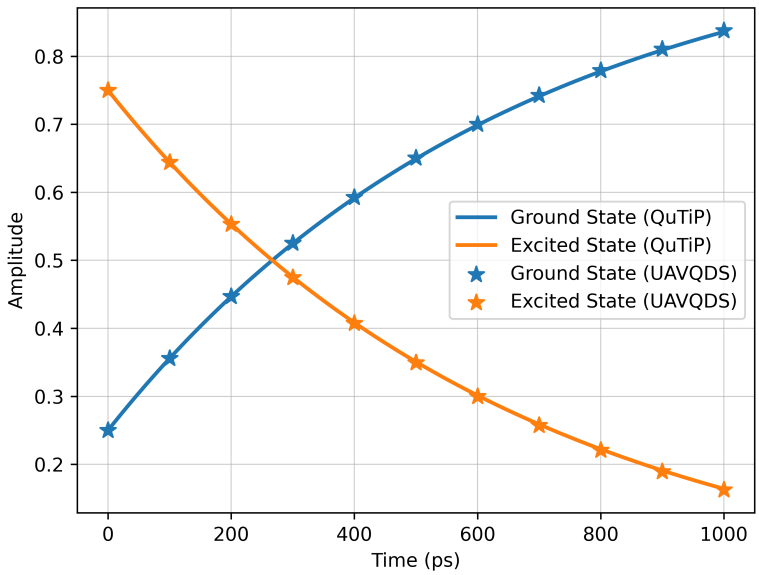

Figure 3: Time evolution of the populations of states $|0\rangle$ and $|1\rangle$ for the amplitude-damping model. Solid lines correspond to the exact QuTiP solution, while star markers denote the results obtained using the UAVQD method.

4 Stochastic Schrödinger Equation Based Variational Approach

The algorithm, showcased in the previous section, uses a vectorized Hamiltonian, is very efficient for simulating the full density matrix of systems with local interactions. However, it has a significant drawback – it requires twice as many qubits. For the near-term devices, the approach discussed in this section is advantageous for the two key reasons. First, the number of qubits needed is the same as the number of qubits required to describe the system's state itself, meaning no extra qubits are necessary. Second, increasing the number of qubits typically demands more entangling gates to form a sufficiently expressive ansatz, which raises the overall gate count and hardware connectivity requirements. In light of these challenges, we showcase an alternative variational algorithm based on the SSE, which offers a way to simulate quantum trajectories more efficiently on near-term quantum hardware.[4,20]

4.1 Stochastic Schrödinger Equation

In the framework of the quantum stochastic Schrödinger equation, the density matrix ρ, which is typically described by the Lindblad master equation, is instead represented by an ensemble of randomly evolving wave functions $\psi_c(t)$, each describing the state of an individual trajectory. Each wave function corresponds to an individual trajectory of the system's evolution. Rather than solving for the full density matrix directly, this method simulates the evolution of pure states under continuous measurement. Importantly, transitioning from the Lindblad equation to the stochastic Schrödinger equation does not involve any approximations. It is simply a different way of expressing the mixed state $\rho(t)$ as a collection of pure states $|\psi(t)\rangle$.[43,44]

Each of these pure state trajectories evolves according to a stochastic Schrödinger equation. For a small time step $\mathrm{d}t$, the change in the wave function $|\psi_c(t)\rangle$ for the specific

trajectory c is given by:

$$\mathrm{d}|\psi_c(t)\rangle = \left[-\mathrm{i}H - \frac{1}{2}\sum_k \left(L_k^\dagger L_k - \langle L_k^\dagger L_k\rangle\right)\right]|\psi_c(t)\rangle\,\mathrm{d}t + \sum_k \left[\frac{L_k|\psi_c(t)\rangle}{\langle L_k^\dagger L_k\rangle} - |\psi_c(t)\rangle\right]\mathrm{d}N_k\,, \quad (4.1)$$

In this equation, the first term represents the smooth, deterministic evolution of the wave function. The second term represents the stochastic part, where the wave function undergoes a sudden change, known as a quantum jump, with a certain probability. These jumps are governed by the jump operators L_k, which describe processes such as excitations or relaxations in the system. The symbol $\langle\cdot\rangle$ denotes the expectation value with respect to the current state of the system.

The term $\mathrm{d}N_k$ is a random variable, it can be either 0 (no jump) or 1 (a jump happens) based on the probability of a quantum jump. The probability that a quantum jump occurs within a small time step dt is given by:

$$\mathrm{d}p = \sum_{k=1}^{K} \frac{\langle\psi(t)|L_k^\dagger L_k|\psi(t)\rangle}{\langle\psi(t)|\psi(t)\rangle}\,\mathrm{d}t\,. \quad (4.2)$$

This formula tells us how likely it is for a jump to happen. If a jump does occur, the state of the system is updated using the operator L_i associated with the jump. If a quantum jump happens, the wave function (state of the system) changes. This new state is calculated as:

$$\tilde{\psi}(t+\mathrm{d}t) = \frac{L_i\tilde{\psi}(t)}{\langle\tilde{\psi}(t)|L_i^\dagger L_i|\tilde{\psi}(t)\rangle}, \quad (4.3)$$

where L_i is randomly chosen from the set of possible jump operators, based on the following probability:

$$p_i = \frac{\langle\tilde{\psi}(t)|L_i^\dagger L_i|\tilde{\psi}(t)\rangle}{\sum_{k=1}^{K}\langle\tilde{\psi}(t)|L_k^\dagger L_k|\tilde{\psi}(t)\rangle}. \quad (4.4)$$

After a jump, we need to normalize the new state, so that it remains a valid quantum state.

The normalized state is:

$$|\psi_j(t)\rangle = \frac{\tilde{\psi}_j(t)}{\langle\tilde{\psi}_j(t)|\tilde{\psi}_j(t)\rangle}, \tag{4.5}$$

where j refers to the specific trajectory.

Finally, to solve the original Lindblad master equation, we use all the individual trajectories to reconstruct the full density matrix. The density matrix is just an average of the outer products of each pure state in the ensemble:

$$\rho(t) = \frac{1}{n}\sum_{j=1}^{n} |\psi_j(t)\rangle\langle\psi_j(t)|, \tag{4.6}$$

where n is the number of trajectories. The more trajectories we include, the more accurately this method describes the evolution of the system.

The Python code simulates quantum system dynamics using a time-stepping algorithm that alternates between deterministic evolution and stochastic quantum jumps. The system evolves deterministically via the effective Hamiltonian until a jump event, determined probabilistically using Lindblad operators, occurs. Upon a jump, a quantum jump operator is randomly selected using `numpy.random` to update the state of the system, and the ansatz parameters are reset accordingly. Because the seed changes between runs, slight variations can appear in the final output. The algorithm records the state, parameters, and jump events over time, capturing the interplay between continuous deterministic evolution and jump-induced quantum transitions.

4.2 Simulation Procedure for Stochastic Evolution

The simulation of SSE begins by preprocessing the input Hamiltonian and Lindblad operators, described in Script S.7.1. These components are decomposed into their Hermitian and anti-Hermitian parts, similar to as discussed in Section 4.1. However, in this approach it is not vectorized so the number of qubit requirements is not be doubled. The Hermitian part drives the system's deterministic evolution, while the anti-Hermitian component incor-

porates the environmental effects introduced by the Lindblad operators. In this step, the Hamiltonian terms are summed to form the Hermitian part, while the anti-Hermitian part is derived from the products of the Lindblad operators and their conjugates.

Next, each trajectory follows the quantum jump algorithm described in Section 4.1. At each time step, the algorithm applies a deterministic evolution unless a quantum jump occurs, at which point a jump operator is applied to the system's state. After every jump, the ansatz parameters are adaptively updated according to the procedure outlined in Section 3.2 and Section 3.3. In Section 3.2, we introduced an unrestricted adaptive procedure that leverages McLachlan's variational principle to tune the parameters of the quantum circuit ansatz at each time step. This ansatz is designed to approximate the evolving quantum state through a series of unitary gates parameterized by the angles $\theta_l(t)$, with the goal of minimizing the McLachlan distance, as shown in Eqs. (3.8) and (3.9). Additionally, Section 3.3 details the construction of the ansatz using a pool of operators. This pool, composed of Pauli operators, allows the algorithm to heuristically select the most appropriate operator to append to the circuit, ensuring accurate modeling of the state while keeping the McLachlan distance below a threshold. These updates to the ansatz at each timestep enable the system to continuously adapt to the noise effects and the stochastic quantum jumps, thereby capturing the system's dynamic evolution effectively.

Once all trajectories are simulated, the final result is obtained by averaging the behavior across all individual trajectories. Averaging over multiple trajectories accounts for the probabilistic nature of quantum jumps, ensuring a comprehensive representation of the system's dynamics by reflecting both the typical evolution and the rare stochastic deviations. This ensemble averaging smooths out random fluctuations, highlighting underlying physical trends in system properties such as energy, population, and coherence. By incorporating the effects of quantum jumps and dissipative interactions, the final averaged result provides a detailed and accurate picture of the open quantum system's overall evolution.

An additional advantage of this approach is that each trajectory can be evolved indepen-

dently, making the simulation highly suitable for parallelization. This method's structure naturally lends itself to parallel processing, particularly useful on multi-core systems. By utilizing Python's built-in multiprocessing package, the evolution of different trajectories can be executed concurrently, significantly improving computational efficiency. This can be achieved by distributing the workload across multiple processors, depending on the user's hardware and operating system. For instance, Python's multiprocessing library allows the execution of trajectories in parallel, as demonstrated in the provided example, Script S.8.1, where multiple quantum trajectories are evolved simultaneously using parallel processes. Users may modify the script and implement parallelization using preferred parallel computing frameworks. This approach can drastically reduce the simulation time, especially when a large number of trajectories are required to obtain statistically accurate results.

4.3 Simulation of the FMO Complex

In Section 2.3, we introduced and discussed the FMO complex example. In this section, we utilize the same example to showcase simulations using a Stochastic Schrödinger Equation (SSE) based method. While we will use the same parameters as before, there is a slight difference in implementation: previously, we initialized every parameter as a QuTiP object, whereas in this Python-based code, we will utilize NumPy arrays.

It's important to note that we pad the arrays to the nearest power of two dimensions suitable for qubit representation. As described in Script S.9.1, the Hamiltonian will be padded from 5×5 to 8×8 and the initial state vector will expand from 5×1 to 8×1.

Once all input parameters have been correctly initialized, you can begin the simulation by submitting the simulation details. To do this, simply import the predefined functions from the `qmad` module. Script S.9.2 is a sample code snippet to illustrate this process. This code sets up the necessary parameters and runs the parallel trajectories for your simulation.

Now that we have completed the preprocessing and processing stages, we will move on to the post-processing phase. As discussed earlier, this quantum simulation approach involves

running simulations over multiple trajectories, each exhibiting distinct stochastic behavior. To obtain the final results, we take the average of the outputs of interest.

The Script S.9.3 illustrates the post-processing procedure for the FMO example, whose results are shown in the corresponding figure. Since high-precision reference dynamics have already been obtained using QuTiP, we directly compare them with the results produced by our method. Overall, the agreement is excellent. As shown in Fig. 4, the stochastic Schrödinger equation (SSE)-based variational simulations closely reproduce the exact QuTiP population dynamics for all five quantum states of the FMO complex, thereby validating the trajectory-based approach.

The small discrepancies that remain originate from controlled numerical and variational approximations, including finite time-step integration, a limited number of stochastic trajectories, truncation of the Ansatz operator pool, and the use of a nonzero threshold parameter in the variational selection. These differences can be systematically reduced by refining these parameters, at the cost of increased computational effort.

5 Conclusions

The simulation of open quantum systems remains a central challenge in quantum computing, particularly in the presence of dissipation, decoherence, and non-unitary dynamics that arise from system-environment interactions. In this installment of the **QFlux** tutorial series, we focused on adaptive variational quantum algorithms as practical and flexible tools for simulating open-system dynamics on near-term quantum hardware.

By introducing `qmad` module to QFlux, we have provided researchers with a simplified toolkit that facilitates the implementation and customization of adaptive variational approaches. The adaptive ansatz methods employed in this work, particularly the unrestricted adaptive variational quantum dynamics (UAVQD) and the stochastic Schrödinger-based algorithm, allow for efficient, flexible circuit structures that can be tailored to the spe-

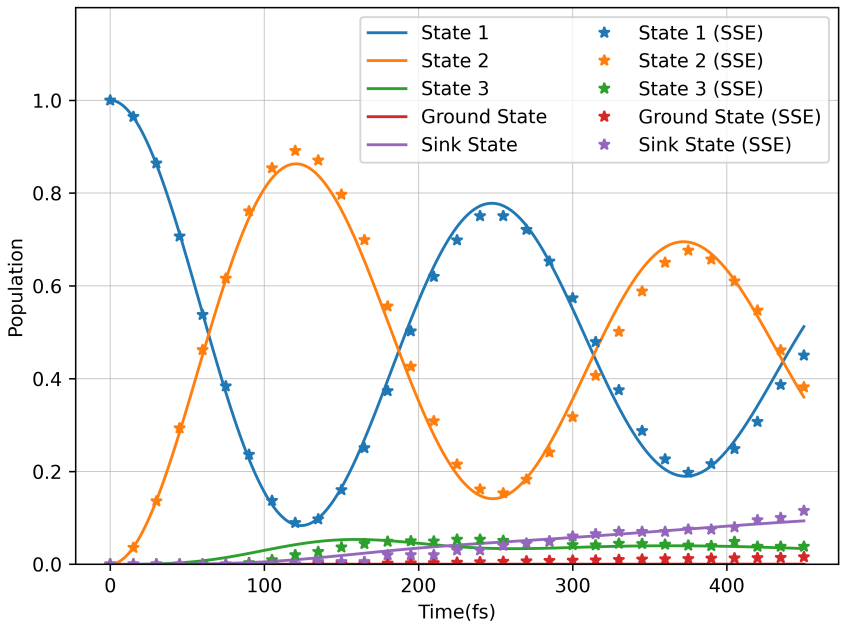

Figure 4: Comparison between exact results obtained using QuTiP (solid lines) and stochastic Schrödinger equation (SSE) based simulations (markers: *) for the population dynamics of five distinct quantum states. Each state's population is represented by a uniquely colored curve.

cific requirements of dissipative processes. Together, these methods illustrate how adaptive strategies can accommodate the diverse requirements of dissipative quantum processes.

All algorithms were implemented within the `qmad` module of the **QFlux** platform, providing an accessible and extensible Python-based environment for adaptive variational simulations. Through representative examples, including the amplitude-damping channel and the Fenna-Matthews-Olson (FMO) complex, we demonstrated that adaptive variational methods can accurately reproduce open-system dynamics while remaining compatible with the constraints of noisy intermediate-scale quantum devices. Benchmarking against exact reference solutions obtained with the QuTiP framework further validated the accuracy and scalability of the proposed approaches.

By translating recent theoretical advances in adaptive variational algorithms into a unified, executable framework, this work lowers the barrier to applying these methods in practice. Together with the preceding parts of the series and the non-Markovian methods de-

veloped in Part VI, this installment establishes adaptive variational quantum algorithms as a key component of the QFlux workflow for simulating realistic open quantum dynamics across a wide range of physical systems.

Supporting Information

Detailed code snippets are available in the Supporting Information and corresponding Google Colab notebook as well as through the QFlux Documentation site.

Acknowledgements

This work was supported by the National Science Foundation under Award No. 2124511 (CCI Phase I: NSF Center for Quantum Dynamics on Modular Quantum Devices, CQD-MQD) and Award No. 2302908 (Engines Development Award: Advancing Quantum Technologies, CT). The authors also acknowledge the use of IBM Quantum services and open-source software packages, including Qiskit, Bosonic Qiskit, Strawberry Fields, QuTiP, and MPSQD.

References

(1) Hu, Z.; Xia, R.; Kais, S. A Quantum Algorithm for Evolving Open Quantum Dynamics on Quantum Computing Devices. *Scientific Reports* **2020**, *10*, 1–9.

(2) Han, J.; Cai, W.; Hu, L.; Mu, X.; Ma, Y.; Xu, Y.; Wang, W.; Wang, H.; Song, Y. P.; Zou, C.-L.; Sun, L. Experimental Simulation of Open Quantum System Dynamics via Trotterization. *Physical Review Letters* **2021**, *127*, 020504.

(3) Wang, H.; Ashhab, S.; Nori, F. Quantum algorithm for simulating the dynamics of an open quantum system. *Physical Review A* **2011**, *83*, 062317.

(4) Endo, S.; Sun, J.; Li, Y.; Benjamin, S. C.; Yuan, X. Variational Quantum Simulation of General Processes. *Physical Review Letters* **2020**, *125*, 010501.

(5) Yao, Y.-X.; Gomes, N.; Zhang, F.; Wang, C.-Z.; Ho, K.-M.; Iadecola, T.; Orth, P. P. Adaptive Variational Quantum Dynamics Simulations. *PRX Quantum* **2021**, *2*, 030307.

(6) Vu, N. P.; Dong, D.; Dan, X.; Lyu, N.; Batista, V.; Liu, Y. A Computational Framework for Simulations of Dissipative Nonadiabatic Dynamics on Hybrid Oscillator-Qubit Quantum Devices. *Journal of Chemical Theory and Computation* **2025**, *21*, 6258–6279.

(7) Wang, Y.; Mulvihill, E.; Hu, Z.; Lyu, N.; Shivpuje, S.; Liu, Y.; Soley, M. B.; Geva, E.; Batista, V. S.; Kais, S. Simulating Open Quantum System Dynamics on NISQ Computers with Generalized Quantum Master Equations. *Journal of Chemical Theory and Computation* **2023**, *19*, 4851–4862.

(8) Schlimgen, A. W.; Head-Marsden, K.; Sager, L. M.; Narang, P.; Mazziotti, D. A. Quantum Simulation of Open Quantum Systems Using a Unitary Decomposition of Operators. *Physical Review Letters* **2021**, *127*, 270503.

(9) Shivpuje, S.; Sajjan, M.; Wang, Y.; Hu, Z.; Kais, S. Designing Variational Ansatz for Quantum-Enabled Simulation of Non-Unitary Dynamical Evolution-An Excursion into Dicke Superradiance. *Advanced Quantum Technologies* **2024**, 2400088.

(10) Hu, Z.; Head-Marsden, K.; Mazziotti, D. A.; Narang, P.; Kais, S. A General Quantum Algorithm for Open Quantum Dynamics Demonstrated with the Fenna-Matthews-Olson Complex. *Quantum* **2022**, *6*, 726.

(11) Su, H.-Y.; Li, Y. Quantum algorithm for the simulation of open-system dynamics and thermalization. *Physical Review A* **2020**, *101*, 012328.

(12) Suri, N.; Barreto, J.; Hadfield, S.; Wiebe, N.; Wudarski, F.; Marshall, J. Two-Unitary

Decomposition Algorithm and Open Quantum System Simulation. *Quantum* **2023**, *7*, 1002.

(13) Zhang, Y.; Hu, Z.; Wang, Y.; Kais, S. Quantum Simulation of the Radical Pair Dynamics of the Avian Compass. *Journal of Physical Chemistry Letters* **2023**, *14*, 832–837.

(14) Benedetti, M.; Fiorentini, M.; Lubasch, M. Hardware-efficient variational quantum algorithms for time evolution. *Physical Review Research* **2021**, *3*, 033083.

(15) Dan, X.; Geva, E.; Batista, V. S. Simulating Non-Markovian Quantum Dynamics on NISQ Computers Using the Hierarchical Equations of Motion. *Journal of Chemical Theory and Computation* **2025**, *21*, 1530–1546.

(16) Dutta, R.; Cabral, D. G. A.; Lyu, N.; Vu, N. P.; Wang, Y.; Allen, B.; Dan, X.; Cortiñas, R. G.; Khazaei, P.; Schäfer, M.; Albornoz, A. C. C. d.; Smart, S. E.; Nie, S.; Devoret, M. H.; Mazziotti, D. A.; Narang, P.; Wang, C.; Whitfield, J. D.; Wilson, A. K.; Hendrickson, H. P.; Lidar, D. A.; Pérez-Bernal, F.; Santos, L. F.; Kais, S.; Geva, E.; Batista, V. S. Simulating Chemistry on Bosonic Quantum Devices. *Journal of Chemical Theory and Computation* **2024**, *20*, 6426–6441.

(17) Allen, B. C.; Batista, V. S.; Cabral, D. G. A.; Cianci, C.; Dan, X.; Dutta, R.; Geva, E.; Hu, Z.; Kais, S.; Khazaei, P.; Lyu, N.; Mulvihill, E.; Shivpuje, S.; Soudackov, A. V.; Vu, N. P.; Wang, Y.; Wilson, C. QFlux — An Open-Source Python Package for Quantum Dynamics Simulations. https://qflux.batistalab.com, 2025; (accessed: 2025-10-12).

(18) Chen, H.; Lidar, D. A. Hamiltonian open quantum system toolkit. *Communications Physics* **2022**, *5*, 112.

(19) Leong, F. Y.; Ewe, W.-B.; Koh, D. E. Variational quantum evolution equation solver. *Scientific Reports* **2022**, *12*, 10817.

(20) Chen, H.; Gomes, N.; Niu, S.; Jong, W. A. d. Adaptive variational simulation for open quantum systems. *Quantum* **2024**, *8*, 1252.

(21) Haegeman, J.; Cirac, J. I.; Osborne, T. J.; Pizorn, I.; Verschelde, H.; Verstraete, F. Time-Dependent Variational Principle for Quantum Lattices. *Phys. Rev. Lett.* **2011**, *107*, 070601.

(22) Balian, R.; Veneroni, M. Static and dynamic variational principles for expectation values of observables. *Annals of Physics* **1988**, *187*, 29–78.

(23) Preskill, J. Quantum computing in the NISQ era and beyond. *Quantum* **2018**, *2*, 79.

(24) Kandala, A.; Mezzacapo, A.; Temme, K.; Takita, M.; Brink, M.; Chow, J. M.; Gambetta, J. M. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature* **2017**, *549*, 242–246.

(25) McClean, J. R.; Romero, J.; Babbush, R.; Aspuru-Guzik, A. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics* **2016**, *18*, 023023.

(26) Grimsley, H. R.; Economou, S. E.; Barnes, E.; Mayhall, N. J. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nature communications* **2019**, *10*, 3007.

(27) Tang, H. L.; Shkolnikov, V.; Barron, G. S.; Grimsley, H. R.; Mayhall, N. J.; Barnes, E.; Economou, S. E. qubit-adapt-vqe: An adaptive algorithm for constructing hardware-efficient ansätze on a quantum processor. *PRX Quantum* **2021**, *2*, 020310.

(28) Krämer, S.; Plankensteiner, D.; Ostermann, L.; Ritsch, H. QuantumOptics. jl: A Julia framework for simulating open quantum systems. *Computer Physics Communications* **2018**, *227*, 109–116.

(29) Javadi-Abhari, A.; Treinish, M.; Krsulich, K.; Wood, C. J.; Lishman, J.; Gacon, J.;

Martiel, S.; Nation, P. D.; Bishop, L. S.; Cross, A. W.; Johnson, B. R.; Gambetta, J. M. Quantum computing with Qiskit. 2024; https://arxiv.org/abs/2405.08810.

(30) Bergholm, V.; Izaac, J.; Schuld, M.; Gogolin, C.; Ahmed, S.; Ajith, V.; Alam, M. S.; Alonso-Linaje, G.; AkashNarayanan, B.; Asadi, A.; others Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968* **2018**,

(31) Cirq Developers Cirq. 2025; https://zenodo.org/doi/10.5281/zenodo.4062499.

(32) Johansson, J. R.; Nation, P. D.; Nori, F. QuTiP: An open-source Python framework for the dynamics of open quantum systems. *Computer Physics Communications* **2012**, *183*, 1760–1772.

(33) van Rossum, G.; Fred L. Drake, J. Python 3 Reference Manual. Python Software Foundation, 2023; Version 3.x.

(34) Johansson, J. R.; Nation, P. D.; Nori, F. QuTiP 2: A Python framework for the dynamics of open quantum systems. *Computer Physics Communications* **2013**, *184*, 1234–1240.

(35) Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; van der Walt, S. J.; Brett, M.; Wilson, J.; Millman, K. J.; Mayorov, N.; Nelson, A. R. J.; Jones, E.; Kern, R.; Larson, E.; Carey, C. J.; Polat, İ.; Feng, Y.; Moore, E. W.; VanderPlas, J.; Laxalde, D.; Perktold, J.; Cimrman, R.; Henriksen, I.; Quintero, E. A.; Harris, C. R.; Archibald, A. M.; Ribeiro, A. H.; Pedregosa, F.; van Mulbregt, P.; SciPy 1.0 Contributors SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **2020**, *17*, 261–272.

(36) Blankenship, R. E. *Molecular Mechanisms of Photosynthesis*; Blackwell Science, 2002.

(37) Adolphs, J.; Renger, T. How proteins trigger excitation energy transfer in the FMO complex of green sulfur bacteria. *Biophysical Journal* **2006**, *91*, 2778–2797.

(38) Skochdopole, N.; Mazziotti, D. A. Functional subsystems and quantum redundancy in photosynthetic light harvesting. *Journal of Physical Chemistry Letters* **2011**, *2*, 2989–2993.

(39) Valleau, S.; Studer, R. A.; Häse, F.; Kreisbeck, C.; Saer, R. G.; Blankenship, R. E.; Shakhnovich, E. I.; Aspuru-Guzik, A. Absence of Selection for Quantum Coherence in the Fenna–Matthews–Olson Complex: A Combined Evolutionary and Excitonic Study. *ACS Central Science* **2017**, *3*, 1086–1095.

(40) Ishizaki, A.; Fleming, G. R. On the Adequacy of the Redfield Equation and Related Approaches to the Study of Quantum Dynamics in Electronic Energy Transfer. *Journal of Chemical Physics* **2009**, *130*, 234110.

(41) Moix, J.; Wu, J.; Huo, P.; Coker, D.; Cao, J. Efficient energy transfer in light-harvesting systems, III: The influence of the eighth bacteriochlorophyll on the dynamics and efficiency in FMO. *Journal of Physical Chemistry Letters* **2011**, *2*, 3045–3052.

(42) McLachlan, A. D. A variational solution of the time-dependent Schrodinger equation. *Molecular Physics* **1964**, *8*, 39–44.

(43) Carmichael, H. J. Quantum trajectory theory for cascaded open systems. *Physical Review Letters* **1993**, *70*, 2273.

(44) Yip, K. W.; Albash, T.; Lidar, D. A. Quantum trajectories for time-dependent adiabatic master equations. *Physical Review A* **2018**, *97*, 022116.

Supporting Information for

QFlux: An Open-Source Toolkit for Quantum Dynamics Simulations on Quantum Computers.

Part V – Adaptive Variational Quantum Algorithms for Open Quantum Systems

Saurabh Shivpuje,[†] Alexander V. Soudackov,[‡] Xiaohan Dan,[‡] Yuchen Wang,[†] Delmar G. A. Cabral,[‡] Brandon C. Allen,[‡] Zixuan Hu,[†] Ningyi Lyu,[‡] Eitan Geva,[§] Victor S. Batista,[*,‡] Sabre Kais,[*,‖]

[†]*Department of Chemistry, Purdue University, West Lafayette, Indiana 47907, USA*

[‡]*Department of Chemistry, Yale Quantum Institute, Yale University, New Haven, CT 06511, USA*

[§]*Department of Chemistry, University of Michigan, Ann Arbor, MI 48109, USA*

[‖]*Department of Electrical and Computer Engineering, Department of Chemistry, North Carolina State University, Raleigh, North Carolina 27606, USA*

E-mail: victor.batista@yale.edu; s.kais@ncsu.edu

Contents

S.1 Linear Solver for the Lindblad Master Equation

Open quantum systems are often described by the Lindblad master equation, a formalism that captures both coherent evolution generated by a Hamiltonian and incoherent processes induced by the environment. QuTiP provides the `mesolve` function that can be used to integrate the Lindblad equation by efficient implementation of a linear solver. To streamline repeated use of this solver throughout the tutorial, we introduce the function `qutip_prop` that is a convenient wrapper.

`qutip_prop` takes as input the system Hamiltonian, an initial density matrix, a time array for the evolution, a list of collapse operators that specify dissipative processes, and a list of observables to be monitored. It then returns the expectation values of the specified observables as functions of time. This modular structure allows us to compare exact Lindblad dynamics with the variational and stochastic methods presented in later sections.

Script S.1.1: Linear Solver for Exact Solutions

```python
from qutip import mesolve, Qobj

def qutip_prop(H, rho0, time_arr, c_ops, observable):
    """
    First import the mesolve function, which is used to solve master equations, and
     the Qobj class, which is used to represent quantum objects, from the QuTiP
     library.
    - H: Hamiltonian of the system (Qobj).
    - rho0: Initial density matrix (Qobj).
    - time_arr: Time array for dynamic simulation (array).
    - c_ops: List of collapse operators (list of Qobj), can be empty for Liouville
     equation.
    - observable: Operator for which the expectation value is to be calculated (Qobj).
    Returns:
    - expec_vals: List of expectation values of the observable over time.
    """
    result = mesolve(H, rho0, time_arr, c_ops, e_ops=observable)
    return result.expect
```

S.2 Amplitude Damping Channel

The amplitude damping channel is a standard model of irreversible decay from an excited state into a ground state, such as spontaneous emission in a two-level atom. It provides an ideal test case for benchmarking dissipative quantum dynamics since it can be solved analytically as shown in the following subsection.

S.2.1 Analytic solution of the Lindblad equation

This section shows, step by step, how to solve the amplitude-damping (energy-relaxation) dynamics analytically by finding an analytic solution of the Lindblad master equation, and how the result maps to the familiar Kraus (operator-sum) form.

1. Choose the Lindblad model (zero-temperature relaxation). Amplitude damping describes an excited state $|1\rangle$ irreversibly relaxing to the ground state $|0\rangle$. In the Markovian limit this is captured by a single jump operator

$$L = \sigma_- = |0\rangle\langle 1|\,,$$

with rate $\gamma \geq 0$. The Lindblad equation reads

$$\frac{\partial \rho(t)}{\partial t} = -\frac{i}{\hbar}[\mathcal{H}, \rho(t)] + \gamma \left(\sigma_- \rho(t) \sigma_+ - \frac{1}{2}\{\sigma_+ \sigma_-, \rho(t)\} \right), \tag{S.1}$$

where $\sigma_+ = |1\rangle\langle 0|$ and $\{\cdot, \cdot\}$ is the anticommutator. For the standard amplitude-damping channel, one often sets $\mathcal{H} = 0$ (pure dissipation), or takes $\mathcal{H} = \frac{\hbar\omega}{2} Z$ (unitary precession plus damping). We solve both in a way that makes the dissipative part transparent.

2. Write the density matrix in the computational basis. Let

$$\rho(t) = \begin{pmatrix} \rho_{00}(t) & \rho_{01}(t) \\ \rho_{10}(t) & \rho_{11}(t) \end{pmatrix}, \qquad \rho_{10}(t) = \rho_{01}(t)^*, \quad \rho_{00}(t) + \rho_{11}(t) = 1.$$

We will derive coupled ODEs for the matrix elements.

3. Evaluate the dissipator element-by-element. Use the identities

$$\sigma_+\sigma_- = |1\rangle\langle 1|, \qquad \sigma_-\rho\,\sigma_+ = \rho_{11}\ |0\rangle\langle 0|.$$

Plugging into the dissipative term

$$\mathcal{D}[\rho] = \gamma\left(\sigma_-\rho\sigma_+ - \frac{1}{2}\{|1\rangle\langle 1|, \rho\}\right)$$

and reading off components gives

- Population of the excited state:

$$\dot\rho_{11}(t) = -\gamma\rho_{11}(t). \tag{S.2}$$

- Population of the ground state (by trace preservation or directly):

$$\dot\rho_{00}(t) = +\gamma\rho_{11}(t). \tag{S.3}$$

- Coherences:

$$\dot\rho_{01}(t) = -\frac{\gamma}{2}\rho_{01}(t), \qquad \dot\rho_{10}(t) = -\frac{\gamma}{2}\rho_{10}(t). \tag{S.4}$$

If you also include $\mathcal{H} = \frac{\hbar\omega}{2}Z$, the commutator contributes a phase rotation:

$$-\frac{i}{\hbar}[\mathcal{H}, \rho] \ \Rightarrow\ \dot\rho_{01}(t) = -i\omega\,\rho_{01}(t), \quad \dot\rho_{10}(t) = +i\omega\,\rho_{10}(t),$$

while leaving ρ_{00}, ρ_{11} unchanged.

4. Solve the ODEs (closed-form solution). From Eq. Eq. (S.2),

$$\rho_{11}(t) = \rho_{11}(0)\, e^{-\gamma t}. \tag{S.5}$$

Then Eq. Eq. (S.3) and $\rho_{00} = 1 - \rho_{11}$ give

$$\rho_{00}(t) = 1 - \rho_{11}(0)\, e^{-\gamma t} = \rho_{00}(0) + \rho_{11}(0)\left(1 - e^{-\gamma t}\right). \tag{S.6}$$

From Eq. Eq. (S.4),

$$\rho_{01}(t) = \rho_{01}(0)\, e^{-\gamma t/2}, \qquad \rho_{10}(t) = \rho_{10}(0)\, e^{-\gamma t/2}. \tag{S.7}$$

Including the Hamiltonian $\mathcal{H} = \frac{\hbar\omega}{2} Z$ simply adds oscillation:

$$\rho_{01}(t) = \rho_{01}(0)\, e^{-(\gamma/2)t}\, e^{-i\omega t}, \qquad \rho_{10}(t) = \rho_{10}(0)\, e^{-(\gamma/2)t}\, e^{+i\omega t}. \tag{S.8}$$

So the analytic amplitude-damping evolution is

$$\rho(t) = \begin{pmatrix} \rho_{00}(0) + \rho_{11}(0)\,(1 - e^{-\gamma t}) & \rho_{01}(0)\, e^{-(\gamma/2)t}\, e^{-i\omega t} \\ \rho_{10}(0)\, e^{-(\gamma/2)t}\, e^{+i\omega t} & \rho_{11}(0)\, e^{-\gamma t} \end{pmatrix}. \tag{S.9}$$

5. Connect to the Kraus (channel) form and identify $\lambda(t)$. The standard amplitude-damping channel is written as

$$\mathcal{E}_\lambda(\rho) = \sum_{k=0}^{1} M_k\, \rho\, M_k^\dagger, \quad M_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\lambda} \end{pmatrix}, \quad M_1 = \begin{pmatrix} 0 & \sqrt{\lambda} \\ 0 & 0 \end{pmatrix}.$$

Applying this map to $\rho(0) = \begin{pmatrix} p & q \\ q^* & 1-p \end{pmatrix}$ yields

$$\rho(t) = \begin{pmatrix} p + (1-p)\lambda & q\sqrt{1-\lambda} \\ q^*\sqrt{1-\lambda} & (1-p)(1-\lambda) \end{pmatrix}.$$

Match this to the Lindblad solution (S.9) with $p = \rho_{00}(0)$ and $1 - p = \rho_{11}(0)$:

$$1 - \lambda(t) = e^{-\gamma t} \quad \Rightarrow \quad \boxed{\lambda(t) = 1 - e^{-\gamma t}}$$

and the coherence factor becomes

$$\sqrt{1-\lambda(t)} = \sqrt{e^{-\gamma t}} = e^{-(\gamma/2)t},$$

exactly as in the analytic ODE solution. This is the cleanest way to see that the Kraus parameter λ is simply the time-dependent decay probability induced by the Lindblad rate γ.

6. Physical interpretation (what the formulas mean). The solution shows two hallmark features of amplitude damping:

- **Population relaxation:** the excited-state population decays exponentially, $\rho_{11}(t) = \rho_{11}(0)e^{-\gamma t}$, transferring weight to $|0\rangle$.
- **Coherence decay:** off-diagonal terms shrink as $e^{-(\gamma/2)t}$ (and rotate at frequency ω if $\mathcal{H} \propto Z$).

At long times $t \to \infty$, $\rho(t) \to |0\rangle\langle 0|$ regardless of the initial state, as expected for zero-temperature relaxation.

S.2.2 Numerical Solution

In this section, we solve the resulting Lindblad master equation using QuTiP, after constructing the lowering operator, specifying a Hamiltonian, defining a normalized initial state, and assembling the collapse operators representing the amplitude damping process.

By monitoring the populations of the ground and excited states, we observe the characteristic exponential relaxation toward the ground state, discussed in Sec. S.2.1. These exact results will later serve as a benchmark for the variational solver and the stochastic unraveling methods.

Script S.2.1 simulates the time evolution of a two level quantum system under amplitude damping using the Lindblad master equation formalism in QuTiP. It defines the system Hamiltonian, collapse operator, initial quantum state, and then numerically solves the open system dynamics over a specified time range. Finally, it extracts and plots the ground and excited state populations as functions of time to visualize relaxation due to damping.

Script S.2.1: Amplitude Damping Channel (Linear Solver)

```python
import numpy as np
import matplotlib.pyplot as plt
from qutip import *

# Pauli matrices and lowering operator
sx = np.array([[0, 1], [1, 0]])
sy = np.array([[0, -1j], [1j, 0]])
sp = (sx + 1j * sy) / 2
sm = Qobj(sp)  # collapse operator for amplitude damping

# Identity Hamiltonian (trivial in this case)
H = Qobj(np.eye(2, dtype=np.complex128))

# Time scale
tf = 1000e-12  # final time
dt = 1e-12  # time step
qt_times = np.arange(0, tf, dt)  # time array

# Amplitude damping rate
gamma = 1.52e9  # damping rate
```

```
# Initial state (normalize if necessary)
u0 = np.array([1 / 2, np.sqrt(3) / 2], dtype=np.complex128)
u0 = u0 / np.linalg.norm(u0)  # normalize the state
psi0 = Qobj(u0)  # initial state as Qobj
rho0 = psi0 * psi0.dag()

# Collapse operators
c_ops = [np.sqrt(gamma) * sm]  # amplitude damping

# Define projectors for ground and excited states
proj_excited = basis(2, 1) * basis(2, 1).dag()  # |1><1|
proj_ground = basis(2, 0) * basis(2, 0).dag()   # |0><0|

# Solve the master equation using mesolve
result = qutip_prop(H, rho0, qt_times, c_ops, [proj_ground, proj_excited])

# Extract ground and excited state populations
ground_population = result[0]  # <0|rho|0> (ground state population)
excited_population = result[1]  # <1|rho|1> (excited state population)

# Plotting results
line1 = plt.plot(qt_times * 1e12, ground_population, linewidth=2, label="Ground
    State Population (QuTiP)")
line2 = plt.plot(qt_times * 1e12, excited_population, linewidth=2, label="Excited
    State Population (QuTiP)")

# Extract colors from the lines
color1 = line1[0].get_color()
color2 = line2[0].get_color()

# Add points every 100 ps
point_indices = range(0, len(qt_times), 100)
#plt.scatter(qt_times[point_indices] * 1e12, ground_population[point_indices],
    color=color1, s=20, zorder=5)
#plt.scatter(qt_times[point_indices] * 1e12, excited_population[point_indices],
    color=color2, s=20, zorder=5)

# Add grid lines every 100 ps on x-axis and every 0.1 on y-axis
plt.xticks(np.arange(0, 1100, 100))
plt.yticks(np.arange(0.1, 0.9, 0.1))
plt.grid(True, which='major', linestyle='-', linewidth=0.5, alpha=0.7)

plt.xlabel('Time (ps)')
plt.ylabel('Population')
plt.legend(loc='center right')
plt.title('Exact Evolution of Ground and Excited State Populations')
plt.show()
```

S.3 FMO Complex

The Fenna–Matthews–Olson (FMO) complex is an important model system in studies of excitonic energy transfer, particularly in the context of quantum biology. It consists of several coupled chromophoric sites through which an excitation migrates before being funneled toward a reaction center. The transport dynamics arise from the interplay of coherent couplings between sites and environmental interactions such as dephasing and irreversible decay. Here, we construct a reduced five-site Hamiltonian for the FMO complex and a corresponding set of Lindblad operators representing various environmental effects. These operators include site-dependent dephasing and an irreversible sink, both of which are crucial for modeling realistic transport pathways. Once the Hamiltonian and dissipators are defined, we evaluate the population dynamics across the network by computing expectation values of projectors onto each site. This allows us to visualize transport processes as a function of time.

This script defines the Hamiltonian and environmental interaction parameters for a five site FMO like quantum system using QuTiP objects. It specifies three types of Lindblad operators corresponding to different dissipation and transfer processes, controlled by the rates alpha, beta, and gamma. Together, these operators characterize how energy relaxation, site dependent noise, and irreversible population transfer affect the system dynamics.

Script S.3.1: FMO Complex parameters

```python
from qutip import Qobj
import numpy as np
# Hamiltonian (Unit: eV)
H = Qobj([
    [0, 0, 0, 0, 0],
    [0, 0.0267, -0.0129, 0.000632, 0],
    [0, -0.0129, 0.0273, 0.00404, 0],
    [0, 0.000632, 0.00404, 0, 0],
```

```
    [0, 0, 0, 0, 0],
])

alpha, beta, gamma = 3e-3, 5e-7, 6.28e-3 #Unit: fs (femtosecond)

# Define the alpha operators
Llist_f = [Qobj(np.diag([0] * i + [np.sqrt(alpha)] + [0] * (4 - i))) for i in
    range(1, 4)]

# Define the beta operators
Llist_f += [Qobj(np.array([[np.sqrt(beta) if i == 0 and j == k else 0 for j in
    range(5)] for i in range(5)])) for k in range(1, 4)]

# Define the gamma operator
L_temp = np.zeros((5, 5))
L_temp[4, 3] = np.sqrt(gamma)
Llist_f.append(Qobj(L_temp))
```

Following code defines measurement operators to track the population of each site, the ground state, and the sink state in a five level quantum system. It initializes the system in the second excited site and propagates the state in time using the previously defined Hamiltonian and Lindblad operators. Finally, it plots the time dependent populations to visualize how excitation energy redistributes and decays across the system.

Script S.3.2: FMO Complex plot

```
# Measurement operators
Mexp_f = [
    Qobj(np.diag([0, 1, 0, 0, 0])),
    Qobj(np.diag([0, 0, 1, 0, 0])),
    Qobj(np.diag([0, 0, 0, 1, 0])),
    Qobj(np.diag([1, 0, 0, 0, 0])),
    Qobj(np.diag([0, 0, 0, 0, 1]))
]

# Time evolution
times = np.linspace(0.0, 450.0, 2000)
psi0_f = Qobj([[0], [1], [0], [0], [0]])

# Using qutip_propagation function
population = qutip_prop(H, psi0_f, times, Llist_f, Mexp_f)
```

```
# Plotting the results
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
labels = ["State 1", "State 2", "State 3", "Ground State", "Sink State"]
for expec, label in zip(population, labels):
   ax.plot(times, expec, label=label)
ax.set_xlabel('Time(fs)')
ax.set_ylabel('Population')
ax.legend()
plt.show()
```

S.4 Vectorized Effective Hamiltonian

Variational methods for open quantum systems frequently operate in the Liouville-space formalism, where density matrices are vectorized and the master equation becomes a linear differential equation. In this representation, the Liouvillian superoperator acts as an effective non-Hermitian Hamiltonian, combining coherent dynamics generated by the system Hamiltonian with dissipative effects arising from Lindblad operators.

To construct this Liouville-space object, we begin by defining a routine that vectorizes the commutator with a given Hamiltonian. We then build the full effective operator by incorporating the dissipative contributions of the Lindblad terms. The resulting structure is essential for the QMAD variational algorithm, which approximates time evolution in Liouville space using parametrized ansätze.

This script constructs a vectorized representation of an effective Hamiltonian acting in Liouville space by transforming commutators into Kronecker products. It combines the coherent Hamiltonian contribution with dissipative Lindblad terms scaled by the decay rate gamma to form a superoperator acting on vectorized density matrices. The result is packaged into a class that separately stores the Hamiltonian and dissipation components for later use in open system simulations.

Script S.4.1: Vectorized Effective Hamiltonian

```
def vectorize_comm(A):
    # Create an identity matrix with the same dimension as A
    iden = np.eye(A.shape[0])
    # Compute the vectorized commutator [A, .] as the Kronecker product
    return np.kron(iden, A) - np.kron(A.T, iden)

class VectorizedEffectiveHamiltonian_class:
    def __init__(self, He, Ha):
        self.He = He
        self.Ha = Ha

def VectorizedEffectiveHamiltonian(H, gamma, lind):
    # Create an identity matrix with the same dimension as H
    iden = np.eye(H.shape[0])
    d = H.shape[0]
    # Compute the vectorized commutator for the Hamiltonian H
    vec_H = vectorize_comm(H)
    # Initialize the result matrix with zeros (complex type)
    res = np.zeros((d**2, d**2), dtype=np.complex128)
    # Compute the conjugate of the Lindblad operator
    L_conj = lind.conj()
    L_dagger_L = L_conj.T @ lind
    # Compute the Lindblad contribution to the effective Hamiltonian
    res -= gamma * (np.kron(L_conj, lind) - (np.kron(iden, L_dagger_L) +
     np.kron(L_dagger_L.T, iden)) / 2)
    # Return an instance of the VectorizedEffectiveHamiltonian_class with vec_H and
     res
    return VectorizedEffectiveHamiltonian_class(vec_H, res)
```

S.5 Operator Pool for Amplitude Damping Channel

Variational algorithms require a flexible and expressive pool of operators from which trial updates to the quantum state or density matrix can be constructed. In the context of the QMAD approach, we build this pool from tensor products of Pauli operators acting on different subsets of qubits. These operators serve as the building blocks of ansätze used to approximate the evolution of vectorized density matrices.

The code below defines a function that generates such an operator pool, systematically exploring combinations of qubit indices and Pauli matrices. We also provide an `Ansatz`

constructor that assembles the initial state, vectorizes the density matrix, and packages the operator pool for use by the variational solver.

Script S.5.1: Defining pool of operators/gates

```
from itertools import combinations, product
from qflux.variational_methods.qmad.ansatz import PauliOperator

# Define Pauli matrices
sx = np.array([[0, 1], [1, 0]])
sy = np.array([[0, -1j], [1j, 0]])
sz = np.array([[1, 0], [0, 0]])

def build_pool(nqbit):
    pauliStr = ["sx", "sz", "sy"]
    res = []
    # Iterate over combinations of qubit indices and Pauli operators
    for order in range(1, 3):
        for idx in combinations(range(1, nqbit + 1), order):
            for op in product(pauliStr, repeat=order):
                res.append(PauliOperator(op, list(idx), 1, nqbit))
    return res
```

S.6 Variational Simulation of Amplitude Damping

Having established the exact amplitude damping dynamics using a linear solver, we now apply the QMAD variational algorithm to approximate the same evolution in Liouville space. This method represents the density matrix as a vector and simulates its trajectory using a parametrized ansatz. The variational parameters are updated by minimizing the residual of the vectorized master equation, enabling an efficient approximation of open-system dynamics.

We construct in this script the vectorized effective Hamiltonian for the amplitude damping channel and initialize a suitable ansatz based on the operator pool defined earlier. We

propagate the system over time using the QMAD solver and directly compare the variational dynamics against the numerically exact results obtained with QuTiP. This comparison highlights both the accuracy and the potential computational advantages of variational approaches for simulating dissipative quantum processes.

Script S.6.1: Amplitude Damping Channel (UAVQDS)

```python
import numpy as np
from qflux.variational_methods.qmad.solver import solve_avq_vect
from qflux.variational_methods.qmad.effh import VectorizedEffectiveHamiltonian
from qflux.variational_methods.qmad.ansatz import Ansatz
import matplotlib.pyplot as plt

sx = np.array([[0, 1], [1, 0]])
sy = np.array([[0, -1j], [1j, 0]])
sp = (sx + 1j * sy) / 2
Id = np.eye(2, dtype=np.complex128)

#-- build effective Hamiltonian for the vectorized density matrix
H = np.eye(2, dtype=np.complex128)
gamma = 1.52e9
lind = sp
H = VectorizedEffectiveHamiltonian(H, gamma, lind)

#-- initial state to build the initial pure state density matrix
u0 = np.array([1 / 2, np.sqrt(3) / 2], dtype=np.complex128) #it should be normalized
    initial state with 2^n length

#-- simulation time (tf) and time step (dt) (in seconds)
tf = 1000e-12
dt =  10e-12

#-- Initialize the ansatz and propagate the variational parameters and density matrix
ansatz = Ansatz(u0, relrcut=1e-6, vectorized=True)
res = solve_avq_vect(H, ansatz, [0, tf], dt, rk45=False)

#-- Print the optimized ansatz at the end of simulation
print("=" * 60)
print(f"Simulation time: {tf*1e12} ps")
print(f"Time step: {dt*1e12} ps")
print("=" * 60)
print("Final Optimized Ansatz")
```

```
print("=" * 60)
print(f"Number of gates in final ansatz: {len(res.A[-1])}")
#print(f"Final ansatz gates: {res.A[-1]}")
print(f"Number of time steps: {len(res.t)}")
print("=" * 60)

#-- Extract diagonal elements from density matrices at each time step
excited = np.array([res.u[i][1, 1].real for i in range(len(res.u))])
ground = np.array([res.u[i][0, 0].real for i in range(len(res.u))])
times = np.array(res.t)

line_qt_1 = plt.plot(qt_times * 1e12, ground_population, label="Ground State
    (QuTiP)", linewidth=2, color='tab:blue')
line_qt_2 = plt.plot(qt_times * 1e12, excited_population, label="Excited State
    (QuTiP)", linewidth=2, color='tab:orange')
color_qt_1 = line_qt_1[0].get_color()
color_qt_2 = line_qt_2[0].get_color()

point_indices_qt = range(0, len(qt_times), 100)
#plt.scatter(qt_times[point_indices_qt] * 1e12,
    ground_population[point_indices_qt], color=color_qt_1, s=20, zorder=5)
#plt.scatter(qt_times[point_indices_qt] * 1e12,
    excited_population[point_indices_qt], color=color_qt_2, s=20, zorder=5)

point_indices = range(0, len(times), 10)
plt.scatter(times[point_indices] * 1e12, ground[point_indices],  color=color_qt_1,
    label="Ground State (UAVQDS)",  s=75, marker='*', zorder=5)
plt.scatter(times[point_indices] * 1e12, excited[point_indices], color=color_qt_2,
    label="Excited State (UAVQDS)", s=75, marker='*', zorder=5)

plt.xlabel("Time (ps)")
plt.ylabel("Amplitude")
plt.grid(True, which='major', linestyle='-', linewidth=0.5, alpha=0.7)
plt.gca().xaxis.set_major_locator(plt.matplotlib.ticker.MultipleLocator(200))
plt.gca().yaxis.set_major_locator(plt.matplotlib.ticker.MultipleLocator(0.1))
plt.legend()
plt.show()
```

S.7 Preprocessing Hamiltonians for the SSE Approach

The stochastic Schrödinger equation (SSE) unravels Lindblad dynamics into an ensemble of pure-state trajectories, whose average reproduces the density-matrix evolution. To employ this approach efficiently, it is helpful to separate the generator into its Hermitian (coherent)

and anti-Hermitian (dissipative) parts and to precompute terms like $L^\dagger L$ that appear in the non-unitary evolution between quantum jumps.

The routine below builds a compact data structure containing: (i) the total Hermitian part, (ii) the effective anti-Hermitian contribution proportional to the sum of $L^\dagger L$, (iii) the full list of Lindblad operators, and (iv) the corresponding $L^\dagger L$ factors. This object is then passed to the trajectory solver to define both drift and jump processes.

Script S.7.1: Preprocessing Hamiltonian for SSE approach

```
class EffectiveHamiltonian_class:
    def __init__(self, He, Ha, Llist, LdL):
        self.He = He  # Hermitian part
        self.Ha = Ha  # Anti-Hermitian part
        self.Llist = Llist  # List of Lindblad operators
        self.LdL = LdL  # List of L^\dagger L

def EffectiveHamiltonian( mats, Llist):
    """
    Create an EffectiveHamiltonian object based on provided parameters.
    :param mats: List of matrices (Hamiltonian terms).
    :param Llist: List of lists of Lindblad operators.
    :return: An instance of EffectiveHamiltonian_class.
    """
    He = sum(mats)  # Sum of Hamiltonian terms as Hermitian part
    Ha = 0.0  # Initialize anti-Hermitian part
    LdL = []  # Initialize the list for Lindblad operator products

    for  LL in Llist:
        for L in LL:
            L_dagger_L = (L.conj().T @ L)
            LdL.append(L_dagger_L)  # Append to LdL list
            Ha += L_dagger_L  # Sum for the anti-Hermitian part

    # Return the Effective Hamiltonian object
    return EffectiveHamiltonian_class(He, 0.5 * Ha, [L for LL in Llist for L in LL],
     LdL)
```

S.8 Parallel Processing of Trajectories

Each SSE trajectory is statistically independent, which makes the method embarrassingly parallel. By distributing trajectory computations across CPU cores, one can reduce wall-clock time and obtain accurate ensemble averages with large sample sizes. The helper function below launches a pool of workers to compute batches of trajectories concurrently and then collects the results. This strategy is particularly useful for realistic models that require hundreds or thousands of trajectories for convergence.

This script defines a helper function to run multiple independent quantum trajectories with identical parameters. It uses Python's multiprocessing Pool to distribute the trajectory simulations across available CPU cores in parallel. The results from all trajectories are collected and returned as a single list for further averaging or analysis.

Script S.8.1: Parallel processing of trajectories

```
from multiprocessing import Pool
def run_trajectories(num_trajectory, H, ansatz, tf, dt):
    # Create a list of tuples with the required parameters for each trajectory
    param_list = [(H, ansatz, tf, dt) for _ in range(num_trajectory)]

    with Pool() as pool:
        results = pool.starmap(solve_avq_trajectory, param_list)

    return results
```

S.9 SSE Simulations of the FMO Complex

We now apply the SSE-based QMAD approach to the FMO complex introduced earlier. To accommodate the variational solver's Hilbert-space requirements, we first pad the FMO Hamiltonian and Lindblad operators to a higher-dimensional space. We then prepare an initial excitation localized on a selected site. This setup allows the algorithm to simulate

dissipative energy transfer using an ensemble of quantum trajectories.

This script prepares the Hamiltonian and dissipation operators for an FMO complex by embedding the original five site system into a larger eight dimensional Hilbert space. It defines multiple Lindblad operators corresponding to different physical processes using the parameters alpha, beta, and gamma, and pads them to match the extended system size. Finally, it initializes the quantum state with a single excitation localized on the second site of the FMO complex.

Script S.9.1: Setting up parameters for FMO Complex

```
#Hamiltonian
H = [
    [0, 0, 0, 0, 0],
    [0, 0.0267, -0.0129, 0.000632, 0],
    [0, -0.0129, 0.0273, 0.00404, 0],
    [0, 0.000632, 0.00404, 0, 0],
    [0, 0, 0, 0, 0],
]
H_fmo= np.pad(H, ((0, 3), (0, 3)), mode='constant')

alpha, beta, gamma = 3e-3, 5e-7, 6.28e-3

# Define the alpha operators
Llist_f = [(np.diag([0] * i + [np.sqrt(alpha)] + [0] * (4 - i))) for i in range(1, 4)]

# Define the beta operators
Llist_f += [(np.array([[np.sqrt(beta) if i == 0 and j == k else 0 for j in range(5)]
    for i in range(5)])) for k in range(1, 4)]

# Define the gamma operator
L_temp = np.zeros((5, 5))
L_temp[4, 3] = np.sqrt(gamma)
Llist_f.append(L_temp)
Llist_f_padded = [np.pad(matrix, ((0, 3), (0, 3)), mode='constant') for matrix in
    Llist_f]

#initial state
u0_fmo = np.zeros(8,dtype=np.complex128)
u0_fmo[1] = 1
```

Following script configures and runs a trajectory based variational quantum dynamics simulation for the FMO complex. It sets the time parameters and number of stochastic trajectories, constructs the effective Hamiltonian from the system Hamiltonian and Lindblad operators, and initializes the variational ansatz from the chosen initial state. Finally, it launches multiple trajectories in parallel to generate an ensemble of quantum evolution results.

Script S.9.2: Setting up parameters for trajectory method

```
from qflux.variational_methods.qmad.solver import solve_avq_trajectory
from qflux.variational_methods.qmad.effh import EffectiveHamiltonian
from qflux.variational_methods.qmad.ansatz import Ansatz

# Guard for multiprocessing in Jupyter
if __name__ == "__main__":
    # Define your parameters (these are placeholders)
    tf = 450        # Final time
    dt = 5          # Time step
    num_trajectory = 200  # Number of trajectories
    H = EffectiveHamiltonian([H_fmo], [Llist_f_padded])  # Initialize the effective
     Hamiltonian
    ansatz = Ansatz(u0_fmo, relrcut=1e-5, vectorized=False)  # Create an Ansatz
     instance

    # Running the parallel trajectories
    results = run_trajectories(num_trajectory, H, ansatz, tf, dt)
```

To extract physical observables from these trajectories, we compute expectation values of diagonal projectors corresponding to excitations on each of the FMO sites and the sink. By averaging over all trajectories, we reconstruct the population dynamics predicted by the Lindblad equation. The result provides insight into the flow of excitation energy across the FMO network.

Script S.9.3: Plot for FMO Complex simulation output

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the observables as diagonal matrices for expectation value calculations
Mexp_f = [
    np.diag([0, 1, 0, 0, 0, 0, 0, 0]),  # Observable 1
    np.diag([0, 0, 1, 0, 0, 0, 0, 0]),  # Observable 2
    np.diag([0, 0, 0, 1, 0, 0, 0, 0]),  # Observable 3
    np.diag([1, 0, 0, 0, 0, 0, 0, 0]),  # Observable 4
    np.diag([0, 0, 0, 0, 1, 0, 0, 0])   # Observable 5
]

# Initialize a list to store the average expectation values
average_expectation_values = []

# Loop over each trajectory to accumulate expectation values
for j in range(num_trajectory):
    # Loop over each observable defined in Mexp_f
    for k, observable in enumerate(Mexp_f):
        expectation_values = []  # List to hold expectation values for the current observable

        # Calculate expectation values for the current trajectory
        for i, psi in enumerate(results[j].psi):
            psi_dagger = np.conjugate(psi).T  # Conjugate transpose of the wave function
            rho = np.outer(psi, psi_dagger)    # Calculate the density matrix
            expectation_value = np.trace(np.dot(rho, observable))  # Compute the expectation value

            # Store the real part of the expectation value
            expectation_values.append(expectation_value.real)

        # Accumulate expectation values for averaging later
        if len(average_expectation_values) <= k:
            average_expectation_values.append(np.array(expectation_values))  # Initialize if not already done
        else:
            average_expectation_values[k] += np.array(expectation_values)  # Sum the values for this observable

# Convert time to femtoseconds for plotting (assuming results[0].t contains time data)
results_t_converted = [t for t in results[0].t]

# Average the accumulated expectation values over all trajectories
average_expectation_values = [ev / num_trajectory for ev in average_expectation_values]
```

```
# QuTiP time evolution
qutip_times = np.linspace(0.0, 450.0, 2000)
qutip_population = population

labels = ["State 1", "State 2", "State 3", "Ground State", "Sink State"]
colors = ["tab:blue", "tab:orange", "tab:green", "tab:red", "tab:purple"]

counter = 0
for expec, label in zip(population, labels):
   plt.plot(qutip_times, expec, label=label, color=colors[counter])
   counter += 1

sse_labels = ["State 1 (SSE)", "State 2 (SSE)", "State 3 (SSE)", "Ground State
    (SSE)", "Sink State (SSE)"]

average_expectation_values = [np.asarray(arr) for arr in
    average_expectation_values]
results_t_converted = np.asarray(results_t_converted)
point_indices = np.arange(0, len(results_t_converted), 3)

counter = 0
for expec, label in zip(average_expectation_values, sse_labels):
   plt.plot(results_t_converted[point_indices], expec[point_indices], '*',
    label=label, color=colors[counter])
   counter += 1
plt.xlabel('Time(fs)')
plt.ylabel('Population')
plt.xticks(np.arange(0, 450, 100))
plt.yticks(np.arange(0, 1.2, 0.2))
plt.grid(True, which='major', linestyle='-', linewidth=0.5, alpha=0.7)
plt.legend(loc='upper right', ncol=2)
plt.ylim(0, 1.2)
plt.tight_layout()
plt.show()
```