30 January 2026

QFlux: An Open-Source Toolkit for Quantum Dynamics Simulations on Quantum Computers. Part VI - The Generalized Quantum Master Equation

Xiaohan Dan[1], Pouya Khazaei[2], Brandon C Allen[1], Ningyi Lyu[1], Callie Wilson[2], Ellen Mulvihill[1], Yuchen Wang[3], Saurabh Shivpuje[3], Sabre Kais[4], Victor S Batista[5,1], Eitan Geva[2]

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

Abstract

Simulating quantum dynamics at finite temperature in complex chemical systems remains a central challenge in quantum chemistry and materials science. In many cases, it is advantageous to focus on the dynamics of a subsystem of interest, represented by its reduced density matrix, whose interaction with the surrounding environment gives rise to open-system behavior. The Nakajima-Zwanzig generalized quantum master equation (GQME) provides a formally exact framework for capturing such non-Markovian dynamics, where memory effects play a defining role. Here, in Part VI of the QFlux tutorial series, we explore methods for simulating non-Markovian open quantum systems described by the GQME on quantum computers. The approach leverages the Sz.-Nagy dilation theorem to embed memory-dependent, non-unitary evolution into an enlarged unitary space suitable for quantum simulation. Using the spin-boson model as a prototypical example, we demonstrate how to formulate, implement, and analyze GQME dynamics within the QFlux open-source platform. This tutorial provides readers with both the theoretical foundation and practical tools required to study non-Markovian quantum dynamics, bridging fundamental concepts with executable workflows for quantum simulation.

QFlux: An Open-Source Toolkit for Quantum Dynamics Simulations on Quantum Computers. Part VI – The Generalized Quantum Master Equation

Xiaohan Dan,[†,⊥] Pouya Khazaei,[‡,⊥] Brandon C. Allen,[†] Ningyi Lyu,[†] Callie Wilson,[‡] Ellen Mulvihill,[†] Yuchen Wang,[¶] Saurabh Shivpuje,[¶] Sabre Kais,[§] Victor S. Batista,[*,†,‖] and Eitan Geva[*,‡]

†*Department of Chemistry, Yale University, New Haven, CT 06520, USA*

‡*Department of Chemistry, University of Michigan, Ann Arbor, MI 48109, USA*

¶*Department of Chemistry, Purdue University, West Lafayette, IN 47907, USA*

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

‖*Yale Quantum Institute, Yale University, New Haven, CT 06511, USA*

⊥*Contributed equally to this work*

E-mail: victor.batista@yale.edu; eitan@umich.edu

Abstract

Simulating quantum dynamics at finite temperature in complex chemical systems remains a central challenge in quantum chemistry and materials science. In many cases, it is advantageous to focus on the dynamics of a subsystem of interest, represented by its reduced density matrix, whose interaction with the surrounding environment gives rise to open-system behavior. The Nakajima–Zwanzig generalized quantum master equation (GQME) provides a formally exact framework for capturing such non-Markovian dynamics, where memory effects play a defining role. Here, in **Part VI** of the **QFlux** tutorial series, we explore methods for simulating non-Markovian open quantum systems described by the GQME on quantum computers. The approach leverages the Sz.-Nagy dilation theorem to embed memory-dependent, non-unitary evolution into an enlarged unitary space suitable for quantum simulation. Using the spin–boson model as a prototypical example, we demonstrate how to formulate, implement, and analyze GQME dynamics within the **QFlux** open-source platform. This tutorial provides readers with both the theoretical foundation and practical tools required to study non-Markovian quantum dynamics, bridging fundamental concepts with executable workflows for quantum simulation.

1 Introduction

In chemistry, biology, and materials science, many dynamical processes unfold in complex environments whose influence cannot be treated as instantaneous or memoryless. Electron and exciton transfer in solution, charge separation in photosynthetic complexes, vibrational relaxation in molecular systems, spin relaxation in superconducting qubits, and phonon-assisted transport in semiconductors are all examples in which the environment retains a persistent memory of past interactions.[1–8] These phenomena arise from intricate couplings between a quantum subsystem and its surrounding bath, giving rise to behavior collectively described as *non-Markovian dynamics*.

In a Markovian regime, environmental effects are assumed to act instantaneously: bath correlations decay rapidly, and the system's evolution depends only on its current state. This approximation underlies the Lindblad quantum master equation introduced earlier in the **QFlux**[9] tutorial series, which provides a practical and widely used framework for modeling dissipative and decoherent dynamics with memoryless reservoirs. Realistic condensed-phase environments, however, often exhibit finite correlation times, structured spectral densities, and temperature-dependent fluctuations that violate the Markovian assumption.[10,11] Accurately describing such systems therefore requires retaining information about how the system's past evolution influences its future behavior.

This installment, **Part VI**, addresses these challenges by focusing on *non-Markovian open-system dynamics* within the framework of the Nakajima–Zwanzig generalized quantum master equation (GQME).[12–14] The GQME provides a formally exact equation of motion for the reduced density operator of the system, introducing a *memory kernel* that captures the delayed influence of the environment. This formulation naturally encompasses a broad range of physical effects, including slow solvent relaxation, vibronic coupling, and long-lived coherence in biological and solid-state systems, and establishes a rigorous connection between microscopic system–bath models and emergent non-Markovian behavior.

Simulating memory-dependent dynamics presents both conceptual and computational challenges. Unlike the Lindblad equation, which is local in time, the GQME is integro-differential: the instantaneous rate of change of the system depends explicitly on its entire prior history. In addition, quantum circuits are inherently unitary, whereas non-Markovian reduced dynamics are fundamentally non-unitary. Bridging this gap requires reformulating the open-system evolution in a form compatible with quantum hardware.[15,16] Recent work has shown that non-Markovian processes described by the GQME can be represented as unitary dynamics in an enlarged Hilbert space using the *Sz.-Nagy dilation theorem*.[7] This construction embeds irreversible reduced dynamics into a higher-dimensional, reversible evolution that can be implemented naturally on qubit-based platforms.

The purpose of this tutorial is to guide readers through the formulation and simulation of non-Markovian quantum dynamics within the **QFlux** framework. We begin by introducing the theoretical structure of the GQME and its physical interpretation in terms of memory effects and environmental back-action. We then show how the Sz.-Nagy dilation formalism enables quantum simulation of these dynamics using unitary circuits acting on an extended system–ancilla register. The spin–boson model serves as a representative case study, illustrating each step of the workflow – from constructing the Liouvillian and memory kernel to executing non-Markovian time evolution on quantum backends.

Part VI thus completes the progression of the **QFlux** tutorial series by extending quantum simulation techniques from closed and Markovian open systems to fully non-Markovian regimes. Together with the earlier parts, this installment equips readers with a comprehensive toolkit for modeling finite-temperature, memory-dependent quantum processes in realistic molecular and materials environments, and for assessing when non-Markovian treatments are essential for capturing the underlying physics.

2 Theory of Generalized Quantum Master Equation

We consider a molecular system described by the Hamiltonian

$$\hat{H} = \sum_{j=1}^{N_e} \hat{H}_j(\hat{\mathbf{R}}, \hat{\mathbf{P}}) \, |j\rangle\langle j| + \sum_{\substack{j,k=1 \\ k\neq j}}^{N_e} \hat{V}_{jk}(\hat{\mathbf{R}}) \, |j\rangle\langle k| \, , \tag{1}$$

where $\hat{H}_j(\hat{\mathbf{R}}, \hat{\mathbf{P}}) = \hat{\mathbf{P}}^2/2 + V_j(\hat{\mathbf{R}})$ is the nuclear Hamiltonian in diabatic state $|j\rangle$, with the index j running over the N_e electronic states. $\hat{V}_{jk}(\hat{\mathbf{R}})$ describes interstate coupling, and $\hat{\mathbf{R}} = \{\hat{R}_1, \hat{R}_2, ..., \hat{R}_{N_n}\}$ and $\hat{\mathbf{P}} = \{\hat{P}_1, \hat{P}_2, ..., \hat{P}_{N_n}\}$ are the mass-weighted nuclear coordinates and momenta of the N_n nuclear degrees of freedom (DOFs). Bold symbols denote vectors, hats indicate operators, and calligraphic letters such as $\mathcal{L}$ represent superoperators.

The evolution of the total density operator $\hat{\rho}(t)$ follows the quantum Liouville equation,

$$\frac{\partial \hat{\rho}(t)}{\partial t} = -\frac{\mathrm{i}}{\hbar}[\hat{H}, \hat{\rho}(t)] = -\frac{\mathrm{i}}{\hbar}\mathcal{L}\hat{\rho}(t)\,, \tag{2}$$

where $\mathcal{L}(\cdot) = [\hat{H}, \cdot]$ is the Liouvillian. Because the Hilbert space of the full electronic-nuclear system scales exponentially with system size, direct solution of Eq. (2) is generally infeasible. In most applications, the focus lies on the electronic subsystem, which serves as the "system" of interest, while nuclear motion acts as a thermal environment or "bath".

The corresponding reduced electronic density operator is

$$\hat{\sigma}(t) = \mathrm{Tr}_n\{\hat{\rho}(t)\} = \sum_{j,k=1}^{N_e} \sigma_{jk}(t)\,|j\rangle\langle k|\,, \tag{3}$$

where Tr_n traces over nuclear degrees of freedom (DOF's). Assuming an initially separable state,

$$\hat{\rho}(0) = \hat{\rho}_n(0) \otimes \hat{\sigma}(0)\,, \tag{4}$$

we seek a closed equation of motion for $\hat{\sigma}(t)$ that incorporates the effects of the bath.

This is achieved through the Nakajima–Zwanzig projection operator formalism.[2,17] To this end, we define the projectors

$$\mathcal{P}\hat{A} = \hat{\rho}_n(0) \otimes \mathrm{Tr}_n\{\hat{A}\}, \qquad \mathcal{Q} = \mathcal{I} - \mathcal{P}, \tag{5}$$

which partition operators into system and bath components. Applying these to Eq. (2) and eliminating the $\mathcal{Q}$ component yields the formally exact generalized quantum master equation (GQME):

$$\frac{\partial \hat{\sigma}(t)}{\partial t} = -\frac{\mathrm{i}}{\hbar}\langle\mathcal{L}\rangle_n^0\,\hat{\sigma}(t) - \int_0^t \mathrm{d}\tau\; \mathcal{K}(\tau)\,\hat{\sigma}(t-\tau)\,, \tag{6}$$

where

$$\langle \mathcal{L} \rangle_n^0(\cdot) = \mathrm{Tr}_n\{\hat{\rho}_n(0)\mathcal{L}\}(\cdot)\,, \tag{7}$$

$$\mathcal{K}(\tau) = \frac{1}{\hbar^2}\,\mathrm{Tr}_n\Big\{\mathcal{L}\,\mathrm{e}^{-\mathrm{i}\mathcal{Q}\mathcal{L}\tau/\hbar}\,\mathcal{Q}\mathcal{L}\,\hat{\rho}_n(0)\Big\}\,. \tag{8}$$

The first term in Eq. (6) represents the instantaneous, mean-field effect of the environment, while the memory kernel $\mathcal{K}(\tau)$ encodes the delayed, history-dependent influence of the bath on the system. When $\mathcal{K}(\tau)$ decays rapidly, the integral term simplifies to a Markovian rate, leading to the Lindblad equation discussed in **Part IV** and **Part V**. When the bath retains long-lived correlations, the full non-Markovian form must be retained.

The forms and derivations of the above GQME, along with its $\langle \mathcal{L} \rangle_n^0$ and $\mathcal{K}(\tau)$, can be found in many previous studies.[17–26] However, the dynamics described by the GQME are inherently non-unitary, as environmental effects induce decoherence and energy dissipation in the system. Recent developments have demonstrated that such non-unitary dynamics can be simulated using the *Sz.-Nagy dilation theorem*, which maps the GQME evolution onto a unitary process within an extended Hilbert space.[7] This transformation enables the simulation of memory-dependent quantum dynamics on gate-based quantum hardware.

In the sections that follow, we demonstrate how to implement this dilation-based simulation within the **QFlux** framework. Using the spin–boson model as an example, we outline the construction of Liouvillians, evaluation of memory kernels, and execution of non-Markovian simulations on classical and quantum backends.

3 The Spin–Boson Model

To illustrate the generalized quantum master equation (GQME) formalism and its quantum simulation, we focus on the spin–boson model—a minimal yet powerful framework for describing electronic energy and charge transfer in molecular and condensed-phase systems.[1,8,27] In this context, the two electronic states ($N_e = 2$) correspond to the diabatic donor and ac-

ceptor states, $|D\rangle$ and $|A\rangle$, respectively. The nuclear degrees of freedom represent a thermal environment that delivers dissipation and decoherence during the transfer process.

Within the spin–boson model, the nuclear Hamiltonians corresponding to $|D\rangle$ and $|A\rangle$ are taken to be harmonic and identical, differing only by a shift in equilibrium geometry and energy. Under these assumptions, the total Hamiltonian reads

$$\hat{H} = \epsilon\,\hat{\sigma}_z + \Gamma\,\hat{\sigma}_x + \sum_{i=1}^{N_n}\left[\frac{\hat{P}_i^2}{2} + \frac{1}{2}\omega_i^2\hat{R}_i^2 - c_i\hat{R}_i\hat{\sigma}_z\right], \tag{9}$$

where $\hat{\sigma}_z = |D\rangle\langle D| - |A\rangle\langle A|$ and $\hat{\sigma}_x = |D\rangle\langle A| + |A\rangle\langle D|$. The parameter 2ϵ defines the energy bias (reaction energy) between donor and acceptor, and $\Gamma = V_{DA}$ denotes the electronic coupling between them. The coupling coefficients c_i describe how each nuclear mode $\hat{R}_i$ interacts with the electronic state difference, while ω_i and $\hat{P}_i$ are the frequency and conjugate momentum of the ith mode.

The influence of the bath is conveniently characterized by its *spectral density*,

$$J(\omega) = \frac{\pi}{2}\sum_{k=1}^{N_n}\frac{c_k^2}{\omega_k}\,\delta(\omega - \omega_k)\,, \tag{10}$$

which quantifies how strongly each frequency mode couples to the system. For concreteness, we adopt the Ohmic spectral density with exponential cutoff,[28–30]

$$J(\omega) = \frac{\pi\hbar}{2}\,\xi\,\omega\,e^{-\omega/\omega_c}\,, \tag{11}$$

where ξ is the Kondo parameter controlling the overall system–bath coupling strength, and ω_c is the cutoff frequency that defines the timescale of bath memory.

Throughout this tutorial, we assume that the system is initially in the donor state and that the nuclear degrees of freedom are in thermal equilibrium with respect to the mean of

the donor and acceptor Hamiltonians. The corresponding initial condition is

$$\hat{\rho}(0) = |D\rangle\langle D| \otimes \hat{\rho}_n(0)\,, \tag{12}$$

$$\hat{\rho}_n(0) = \frac{\exp\left[-\beta \sum_{i=1}^{N_n} \left(\frac{\hat{P}_i^2}{2} + \frac{1}{2}\omega_i^2 \hat{R}_i^2\right)\right]}{\mathrm{Tr}_n\left\{\exp\left[-\beta \sum_{i=1}^{N_n} \left(\frac{\hat{P}_i^2}{2} + \frac{1}{2}\omega_i^2 \hat{R}_i^2\right)\right]\right\}}\,, \tag{13}$$

where $\beta = 1/(k_B T)$ is the inverse temperature. The spin–boson model is therefore fully specified by five parameters: the energy bias ϵ, coupling strength Γ, inverse temperature β, system–bath coupling constant ξ, and cutoff frequency ω_c (Script S.1.1).

The spin–boson model has served as a fundamental testing ground for open-system methodologies for decades, inspiring a wide range of numerically exact and approximate approaches.[6,10,30–39] As a benchmark for the simulations presented here, we employ the numerically exact tensor-train thermofield dynamics (TT-TFD) method,[30] which has been integrated into the **QFlux** package. The derivation of the TT-TFD formalism and details of its implementation were introduced in **Part I**. The TT-TFD propagates the thermal quantum state using the time-dependent variational principle (TDVP)[40] and performs tensor-train (TT) operations efficiently via the **mpsqd** library.[41] The Script S.1.2 demonstrates how to perform such a simulation within **QFlux**.

The TT-TFD results provide a reference for evaluating the performance of quantum simulations based on the GQME and Sz.-Nagy dilation formalism. In the next section, we use the spin–boson model to demonstrate how the projected Liouvillian and memory kernel are constructed within **QFlux**, and how the resulting non-Markovian dynamics can be simulated and compared to the TT-TFD benchmark.

Since the spin-boson model contains two electronic states ($N_e = 2$), the reduced electronic density operator is represented by a 2×2 matrix which can be given in terms of four matrix elements: $\sigma_{ij}(t) = \langle i|\hat{\sigma}(t)|j\rangle$ with $i, j \in \{D, A\}$. Here, the diagonal elements $\sigma_{ii}(t)$, which are known as *populations*, correspond to the occupancies of the donor and acceptor

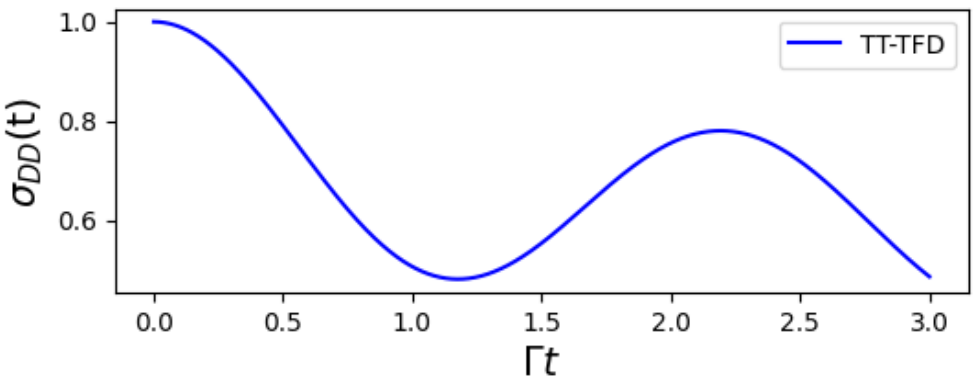

Figure 1: Population dynamics on the donor state $|D\rangle$ for the spin–boson model, obtained using the TT-TFD method. The oscillatory behavior reflects coherent energy exchange between the donor and acceptor states, while the gradual decay arises from bath-induced dissipation.

states, while the off-diagonal terms, which are known as coherences, contain information about the coherent nature of the state.[1] Focusing on electronic energy and charge transfer, our focus would be on the dynamics of the populations of the donor and acceptor states, $\{\sigma_{DD}(t), \sigma_{AA}(t)\}$.

The electronic reduced density operator is represented in the code in its vectorized form:

$$\hat{\sigma}(t) \equiv [\sigma_{DD}(t), \sigma_{DA}(t), \sigma_{AD}(t), \sigma_{AA}(t)]^{\top} \tag{14}$$

Thus, according to Eq. (12), the electronic initial state is $\hat{\sigma}(0) = |D\rangle\langle D| = [1, 0, 0, 0]^{\top}$. In this representation, the super-operators $\langle\mathcal{L}\rangle_n^0$ and $\mathcal{K}(t)$ are represented by 4×4 matrices.

4 The projected Liouvillian

We start with determining $\langle\mathcal{L}\rangle_n^0$ in Eq. (7). This electronic superoperator is defined by the way it acts on an arbitrary electronic operator $\hat{A}$:

$$\langle\mathcal{L}\rangle_n^0 \hat{A} = \mathrm{Tr}_n \left\{ \left[\hat{H}, \hat{\rho}_n(0) \otimes \hat{A}\right] \right\} = \left[\epsilon\hat{\sigma}_z + \Gamma\hat{\sigma}_x, \hat{A}\right] \quad . \tag{15}$$

To get the second equality, we used the property

$$\left[\frac{\hat{P}_i^2}{2}+\frac{1}{2}\omega_i^2\hat{R}_i^2,\hat{\rho}_n(0)\otimes\hat{A}\right]=\left[\frac{\hat{P}_i^2}{2}+\frac{1}{2}\omega_i^2\hat{R}_i^2,\hat{\rho}_n(0)\right]\otimes\hat{A}=0 \quad ,$$

and

$$\mathrm{Tr}_n\left\{c_i\hat{R}_i\hat{\rho}_n(0)\right\}=0 \quad ,$$

since the expectation value of the position operator in an unshifted harmonic oscillator at thermal equilibrium vanishes.

Using the vectorized form of the system subspace operator in Eq. (14), $\langle\mathcal{L}\rangle_n^0$ in Eq. (15) can be written as

$$\langle\mathcal{L}\rangle_n^0=\begin{pmatrix} 0 & -\Gamma & \Gamma & 0 \\ -\Gamma & 2\epsilon & 0 & \Gamma \\ \Gamma & 0 & -2\epsilon & -\Gamma \\ 0 & \Gamma & -\Gamma & 0 \end{pmatrix}. \tag{16}$$

The projected Liouvillian $\langle\mathcal{L}\rangle_n^0$ describes the time evolution that would be observed if the system was uncoupled from the bath [i.e. the memory kernel $\mathcal{K}(t)$ in Eq. (8) is zero since $\mathcal{P}=\mathcal{I}$ and $\mathcal{Q}=0$], integrating Eq. (6) with $\mathcal{K}(t)=0$ gives

$$\hat{\sigma}(t)=\mathrm{e}^{-\frac{\mathrm{i}}{\hbar}\langle\mathcal{L}\rangle_n^0 t}\,\hat{\sigma}(0) \tag{17}$$

The Scripts S.2.1 and S.2.2 show how the dynamics can be calculated and the result is shown in Fig. 2. The system oscillates between the donor state $|D\rangle$ and the acceptor state $|A\rangle$, which corresponds to the time evolution of the pure system without the nuclear bath (i.e. dynamics of the two-level closed system[1]). Compared to the TT-TFD results, coupling to the bath brings in the dissipation effect, which makes the oscillation decay.

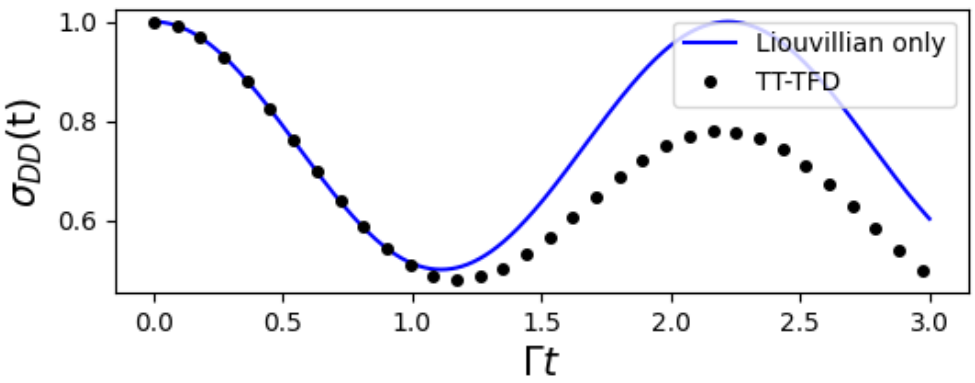

Figure 2: The population dynamics on the donor state $|D\rangle$. Here the dynamics correspond to the projected Liouvillian only. The numerically exact TT-TFD result is also shown for comparison.

5 The memory kernel

The memory kernel, Eq. (8), is obtained by solving the following Volterra equation of the second kind:[6,33,42–44]

$$\mathcal{K}(t) = \mathrm{i}\dot{\mathcal{F}}(t) - \frac{1}{\hbar}\mathcal{F}(t)\langle\mathcal{L}\rangle_n^0 + \mathrm{i}\int_0^t \mathrm{d}\tau\ \mathcal{F}(t-\tau)\,\mathcal{K}(\tau)\ . \tag{18}$$

Here, the projection-free inputs (PFIs) $\mathcal{F}(t)$ and $\dot{\mathcal{F}}(t)$ are

$$\mathcal{F}(t) = \frac{1}{\hbar}\ \mathrm{Tr}_n\Big[\mathcal{L}\mathrm{e}^{-\mathrm{i}\mathcal{L}t/\hbar}\hat{\rho}_n(0)\Big] \tag{19}$$

$$\dot{\mathcal{F}}(t) = -\frac{\mathrm{i}}{\hbar^2}\ \mathrm{Tr}_n\Big[\mathcal{L}\mathrm{e}^{-\mathrm{i}\mathcal{L}t/\hbar}\mathcal{L}\hat{\rho}_n(0)\Big]\ . \tag{20}$$

As shown in Script S.3.1, **QFlux** evaluates the memory kernel directly. One first specifies the spin–boson model to create a `DynamicsGQME` object `SBM`, and uses TT–TFD to compute the propagator (see Section 5.1). Once the propagator is available, invoking the `get_memory_kernel` method of the `DynamicsGQME` class computes the PFIs and then solves Eq. (18) for $\mathcal{K}(t)$.

In the following, we provide a step-by-step illustration of how to calculate the PFIs and

then use them to obtain the memory kernel by solving Eq. (18).

5.1 Calculation of the Projection-Free Inputs

The PFIs may be obtained in multiple ways.[21,24,30,38] In this tutorial, we construct them via TT–TFD. We note that $\mathcal{F}(t) = \mathrm{i}\,\dot{\mathcal{G}}(t)$, where $\mathcal{G}(t)$ is the (in general non-unitary) time-evolution superoperator (propagator), defined by

$$\hat{\sigma}(t) = \mathcal{G}(t)\hat{\sigma}(0) = \mathrm{Tr}_n\left[\mathrm{e}^{-\mathrm{i}\mathcal{L}t/\hbar}\,\hat{\rho}_n(0)\right]\hat{\sigma}(0)\,. \tag{21}$$

Thus, $\mathcal{F}(t)$ and $\dot{\mathcal{F}}(t)$ follow from time-derivatives of the propagator $\mathcal{G}(t)$. The superoperator $\mathcal{G}(t)$ has elements $\mathcal{G}_{jk,lm}(t)$, $j,k,l,m \in \{D,A\}$, obtained by preparing $|l\rangle\langle m| \otimes \hat{\rho}_n(0)$ at t=0 and measuring $|j\rangle\langle k|$ at time t. The Script S.3.2 computes $\{\mathcal{G}_{jk,lm}(t)\}$ using the TT–TFD method.[30]

Once $\{\mathcal{G}_{jk,lm}(t)\}$ are obtained via TT-TFD, the PFIs $\mathcal{F}(t)$ and $\dot{\mathcal{F}}(t)$ can be obtained from it by taking time derivatives (see Script S.3.3).

5.2 Computation of the Memory Kernel

The memory kernel is obtained by solving Eq. (18). We rewrite it as

$$\mathcal{K}(t) = g(t) + \int_0^t f(t-\tau)\,\mathcal{K}(\tau)\,\mathrm{d}\tau\,, \tag{22}$$

where

$$g(t) = \mathrm{i}\dot{\mathcal{F}}(t) - \frac{1}{\hbar}\mathcal{F}(t)\langle\mathcal{L}\rangle_n^0, \qquad f(t-\tau) = \mathcal{F}(t-\tau).$$

This can be implemented as shown in Script S.3.4.

We solve Eq. (22) by fixed-point iteration at discrete times $t_n = n\Delta t$, $n = 0,\ldots,N$ with

$N\Delta t = t$. The iteration is initialized with $\mathcal{K}^{(0)}(t_n) = g(t_n)$ and proceeds until convergence:

$$\begin{aligned}
\mathcal{K}^{(0)}(t_n) &= g(t_n), \\
\mathcal{K}^{(1)}(t_n) &= g(t_n) + \int_0^{t_n} \mathrm{d}\tau \; f(t_n - \tau)\, \mathcal{K}^{(0)}(\tau), \\
\mathcal{K}^{(2)}(t_n) &= g(t_n) + \int_0^{t_n} \mathrm{d}\tau \; f(t_n - \tau)\, \mathcal{K}^{(1)}(\tau), \\
&\vdots \\
\mathcal{K}^{(i)}(t_n) &= g(t_n) + \int_0^{t_n} \mathrm{d}\tau \; f(t_n - \tau)\, \mathcal{K}^{(i-1)}(\tau),
\end{aligned}$$

with stopping criterion

$$\left|\mathcal{K}^{(i)}_{jk,lm}(t_n) - \mathcal{K}^{(i-1)}_{jk,lm}(t_n)\right| \leq 10^{-10} \quad \text{for all } j, k, l, m, n.$$

The time integrals are evaluated using the trapezoidal rule, and convergence is tested element by element for all matrix elements and time steps (Script S.3.5).

The function `CalculateIntegral` in Script S.3.6 calculates the integral part of the Volterra equation through the trapezoidal rule which approximates an integral on a uniform grid with N slices as:

$$\int_a^b f(t)\, \mathrm{d}t \approx h\left[\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{k=1}^{N-1} f(a + kh)\right], \tag{23}$$

where $h = (a - b)\,/\,N$.

As an illustration, Fig. 3 shows two representative elements of the memory kernel, $\mathcal{K}_{DD,DD}(t)$ and $\mathcal{K}_{DA,DD}(t)$. The quantity $\mathcal{K}_{DD,DD}(t)$ remains small, indicating that the environment has only a weak direct effect on the population $\sigma_{DD}(t)$. In contrast, $\mathcal{K}_{DA,DD}(t)$ exhibits a larger amplitude - a bath-induced process that converts the population $\sigma_{DD}(\tau)$ into the coherence $\sigma_{DA}(t)$.

The memory kernel matrix element $\mathcal{K}_{DA,DD}(t)$ starts at zero, rises, and then decays as the bath relaxes. If it decays to zero by a characteristic time τ_B, then the bath loses its memory beyond τ_B, and values of $\sigma_{DD}(t - \tau'_B)$ for $\tau'_B > \tau_B$ no longer contribute to the current dynamics of $\sigma_{DA}(t)$.

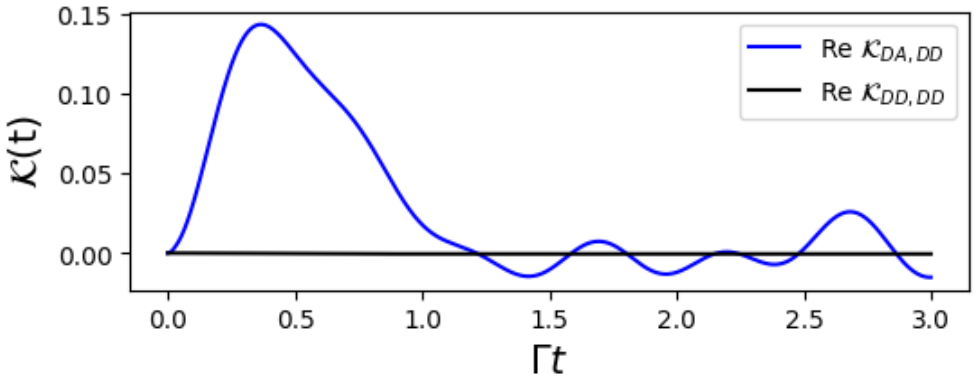

Figure 3: Memory kernel for the Spin-Boson model visualized using Script S.3.7. Here only $\mathcal{K}_{DD,DD}(t)$ and $\mathcal{K}_{DA,DD}(t)$ elements are shown.

6 Solution of the GQME

Given $\langle \mathcal{L} \rangle_n^0$ and $\mathcal{K}(\tau)$ from the preceding subsections, the GQME in Eq. (6) can be numerically integrated using a 4th-order Runge-Kutta (RK4) scheme. In practice, this is done by calling the `solve_gqme` method of the `DynamicsGQME` class in **QFlux**. In Script S.4.1, the memory kernel is supplied together with a chosen memory-time cutoff, and `solve_gqme` returns the time-evolved reduced density operator $\hat{\sigma}(t)$.

In the following Script S.4.2, we detail its implementation. Starting from the initial value problem

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(t, y) \quad \text{with an initial value} \quad y(t_0) = y_0 \ , \tag{24}$$

and substituting $\hat{\sigma}(t)$ for y, the RK4 method propagates y from time t_n to time t_{n+1} ($n =$

$0, 1, 2, ...$) as follows:

$$y_{n+1} = y_n + \frac{h}{6}\Big(k_1 + 2k_2 + 2k_3 + k_4\Big), \tag{25}$$

where $y_n = y(t_n)$, $y_{n+1} = y(t_{n+1})$, h is the time step, and

$$\begin{aligned}
k_1 &= f\Big(t_n, y_n\Big) \\
k_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right) \\
k_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right) \\
k_4 &= f\Big(t_n + h, y_n + hk_3\Big).
\end{aligned}$$

Compared to the GQME in Eq. (6), the time-derivate function $f(t, y)$ in Eq. (24) is

$$f\Big(t, \hat{\sigma}\Big) = -\frac{\mathrm{i}}{\hbar}\sum_{lm}\langle\mathcal{L}_{jk,lm}\rangle_n^0\hat{\sigma}_{lm}(t) - \sum_{lm}\int_0^t \mathrm{d}\tau\, \mathcal{K}_{jk,lm}(\tau)\hat{\sigma}_{lm}(t-\tau) \quad , \tag{26}$$

which is calculated using the extended trapezoidal rule using the function `Calculatef` in Script S.4.3.

With the functions `PropagateRK4` and `Calculatef` defined, the GQME is solved to obtain $\hat{\sigma}(t)$ as implemented in Script S.4.4.

The result shows that the dynamics calculated from GQME are the same as the Exact TT-TFD result which demonstrates the correctness of our memory kernel and GQME approach.

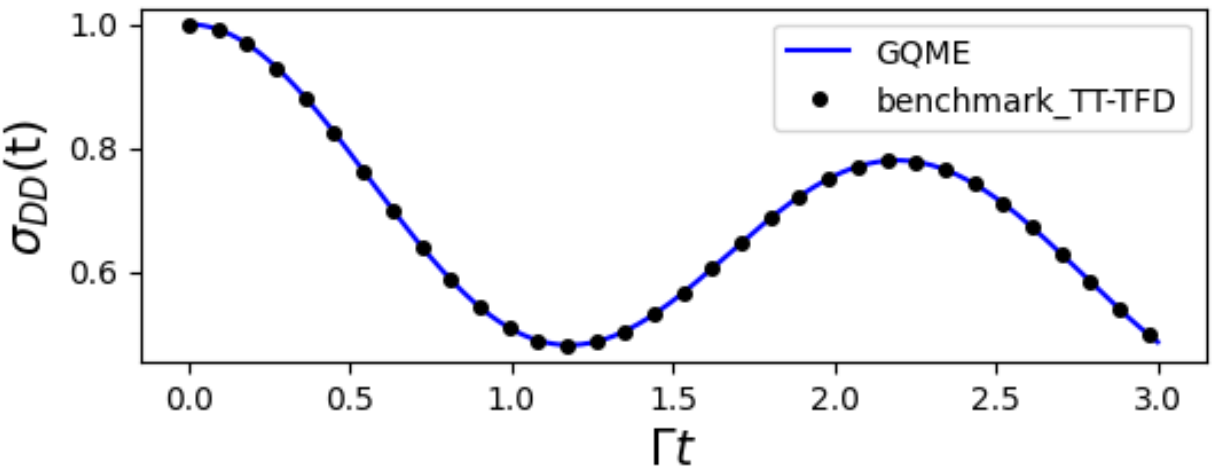

Figure 4: GQME result of the population on the donor state $|D\rangle$ for the Spin-Boson model, compared with numerically exact TT-TFD result.

7 Quantum Algorithms of GQME based on Dilation

7.1 Solving the GQME to get the propagator

In this section we introduce the quantum simulation of the GQME. The reduced dynamics of an open system can be written in terms of its (generally *non-unitary*) propagator $\mathcal{G}(t)$:

$$\hat{\sigma}(t) = \mathcal{G}(t)\,\hat{\sigma}(0)\,. \tag{27}$$

The non-unitarity of $\mathcal{G}(t)$ reflects the coupling to an environment, which induces irreversible processes and memory effects in the subsystem.

Substituting Eq. (27) into Eq. (6) and using the fact that the GQME must hold for an arbitrary $\hat{\sigma}(0)$, one finds that $\mathcal{G}(t)$ satisfies the same generalized quantum master equation:

$$\frac{\partial \mathcal{G}(t)}{\partial t} = -\frac{\mathrm{i}}{\hbar}\,\langle \mathcal{L} \rangle_n^0\,\mathcal{G}(t) - \int_0^t \mathrm{d}\tau\,\mathcal{K}(\tau)\,\mathcal{G}(t-\tau)\,. \tag{28}$$

Starting from the identity superoperator $\mathcal{G}(0) = I$, one can therefore compute $\mathcal{G}(t)$ by solving the same GQME with the $\langle \mathcal{L} \rangle_n^0$ and memory kernel $\mathcal{K}(\tau)$ obtained in the previous sections,

as implemented in Script S.5.1.

7.2 Dilation of the non-unitary propagator

We now introduce the key step enabling quantum simulation: the Sz.-Nagy unitary dilation procedure,[45–48] which embeds the non-unitary propagator $\mathcal{G}(t)$ into a unitary evolution on an enlarged Hilbert space.[7]

We begin by computing the operator norm of $\mathcal{G}(t)$ to determine whether it is a contraction:

$$\|\mathcal{G}(t)\|_O = \sup_{\boldsymbol{v}\neq 0} \frac{\|\mathcal{G}(t)\boldsymbol{v}\|}{\|\boldsymbol{v}\|} \leq 1\,. \tag{29}$$

If $\mathcal{G}(t)$ is not a contraction, we introduce a normalization factor $n_c > \|\mathcal{G}(t)\|_O$ and define the rescaled propagator $\mathcal{G}'(t) = \mathcal{G}(t)/n_c$, which is then a contraction.

For a contraction $\mathcal{G}'(t)$, the Sz.-Nagy unitary dilation is given by

$$\mathcal{U}_{\mathcal{G}'}(t) = \begin{pmatrix} \mathcal{G}'(t) & \mathcal{D}_{\mathcal{G}'^\dagger}(t) \\ \mathcal{D}_{\mathcal{G}'}(t) & -\mathcal{G}'^\dagger(t) \end{pmatrix}, \tag{30}$$

where

$$\mathcal{D}_{\mathcal{G}'}(t) = \sqrt{I - \mathcal{G}'^\dagger(t)\mathcal{G}'(t)}, \qquad \mathcal{D}_{\mathcal{G}'^\dagger}(t) = \sqrt{I - \mathcal{G}'(t)\mathcal{G}'^\dagger(t)}\,. \tag{31}$$

The operator $\mathcal{U}_{\mathcal{G}'}(t)$ is unitary on a Hilbert space of doubled dimension and reproduces the action of $\mathcal{G}'(t)$ on the original space:

$$\mathcal{G}'(t)\hat{\sigma}(0) \quad \xrightarrow{\text{dilation}} \quad \mathcal{U}_{\mathcal{G}'}(t) \begin{pmatrix} \hat{\sigma}(0)^\top \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{32}$$

such that projecting the output back onto the original subspace yields the same result as

applying $\mathcal{G}'(t)$ directly.

The function `dilate` in Script S.5.2 implements this procedure: given $\mathcal{G}(t)$ as input it returns the dilation $\mathcal{U}_{\mathcal{G}'}(t)$ and the corresponding normalization factor n_c.

7.3 Quantum Simulation of GQME with QASM Simulator

In this section, we will delve into the simulation of GQME using Qiskit's QASM simulator, focusing on the spin-boson model.[7] All components of the quantum algorithm are implemented in the QFlux package, as shown in Script S.5.3.

Here are the detailed steps of the implementation:

- the quantum algorithm starts from initializing the quantum circuit with the initial state $\left(\hat{\sigma}(0)^T, 0, \cdots, 0\right)^\top$.
- For the spin-boson model, this requires three qubits with two from the four components of $\hat{\sigma}(0)$ as in Eq. (14) and one from the dilation procedure that doubles the space.
- After initialization, the dilated propagator $\mathcal{U}_{\mathcal{G}'}(t)$ is converted into a quantum gate and applied to the quantum circuit.
- Then, measuring two qubits at 00 or 11 [the first or fourth component in Eq. (14)] with the dilated qubit at 0, the electronic populations can be retrieved by taking the square root of the measuring probability, and multiplying by the normalization factor n_c is the dilation process: $\hat{\sigma}_{DD}(t) = n_c\sqrt{P_{000}}$ and $\hat{\sigma}_{AA}(t) = n_c\sqrt{P_{011}}$.

The corresponding quantum circuit is shown in Figure 5.

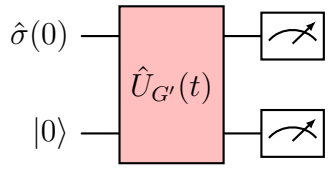

Figure 5: Circuit for implementing the GQME with a one-qubit dilation.

For each specific time t, we generate the quantum circuit and perform the simulations. We implement the quantum circuit using Qiskit's QASM simulator, which is shown in Scripts S.6.1 and S.6.2.

With the QASM simulations complete, we can generate the plots using Script S.6.3 and compare the resulting electronic state population dynamics to the Exact TT-TFD result (see Fig. 6).

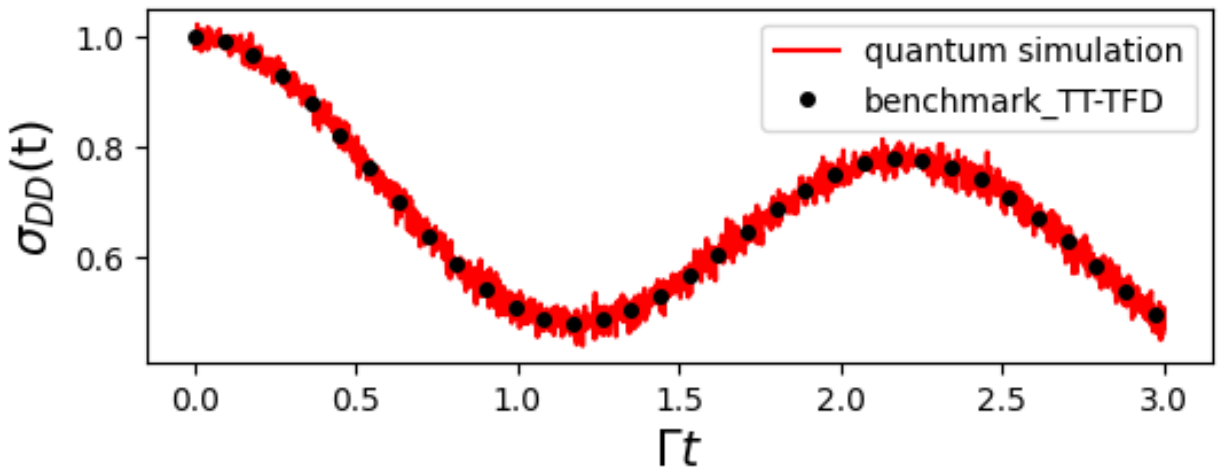

Figure 6: Electronic donor state population dynamics of the spin-boson model, simulated by the GQME-based quantum algorithm as implemented on the IBM QASM quantum simulator. The result is compared to the numerically exact TT-TFD result.

8 Conclusions

This tutorial introduced the generalized quantum master equation (GQME) as a rigorous framework for simulating non-Markovian open quantum dynamics and demonstrated its practical implementation using the spin–boson model as a representative example. We reviewed the formal structure of the GQME, highlighting the role of the memory kernel in capturing delayed environmental effects, and showed how classical reference simulations can be carried out efficiently using Python-based tools.

Building on this foundation, we presented a quantum algorithm based on the Sz.-Nagy

dilation theorem that embeds the inherently non-unitary, memory-dependent GQME dynamics into a unitary evolution on an enlarged Hilbert space. This construction enables non-Markovian dynamics to be simulated within the standard circuit model of quantum computation and provides a clear pathway for benchmarking quantum implementations against classical results.

All elements of the workflow were implemented within the **QFlux** platform, which offers a reproducible and streamlined environment spanning model formulation, classical validation, and quantum execution. Together, these components make this tutorial a practical entry point for the simulation of finite-temperature, non-Markovian open quantum dynamics and establish a foundation for extending quantum simulation methods beyond the Markovian regime on current and emerging quantum hardware.

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), Award CHE 2154114 (Quantum master equations for simulating chemical dynamics), 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) Nitzan, A. *Chemical Dynamics in Condensed Phases. Relaxation, Transfer, and Reactions in Condensed Molecular Systems*; Oxford University Press: New York, 2014.

(2) Breuer, H.-P.; Petruccione, F. *The Theory of Open Quantum Systems*; Oxford University Press: New York, 2007.

(3) Hammes-Schiffer, S.; Stuchebrukhov, A. A. Theory of Coupled Electron and Proton Transfer Reactions. *Chemical Reviews* **2010**, *110*, 6939–6960.

(4) Collini, E.; Wong, C. Y.; Wilk, K. E.; Curmi, P. M. G.; Brumer, P.; Scholes, G. D. Coherently wired light-harvesting in photosynthetic marine algae at ambient temperature. *Nature* **2010**, *463*, 644–647.

(5) Clarke, J.; Wilhelm, F. K. Superconducting quantum bits. *Nature* **2008**, *453*, 1031–1042.

(6) Dan, X.; Xu, M.; Yan, Y.; Shi, Q. Generalized master equation for charge transport in a molecular junction: Exact memory kernels and their high order expansion. *Journal of Chemical Physics* **2022**, *156*, 134114.

(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) 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.

(9) 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).

(10) Dan, X.; Xu, M.; Stockburger, J. T.; Ankerhold, J.; Shi, Q. Efficient low-temperature simulations for fermionic reservoirs with the hierarchical equations of motion method: Application to the Anderson impurity model. *Physical Review B* **2023**, *107*, 195429.

(11) Dan, X.; Long, Z.; Qiu, T.; Menzel, J. P.; Shi, Q.; Batista, V. Nonadiabatic H-Atom Scattering Channels on Ge (111) Elucidated by the Hierarchical Equations of Motion. *arXiv preprint arXiv:2509.16916* **2025**,

(12) Nakajima, S. On Quantum Theory of Transport Phenomena: Steady Diffusion. *Progress of Theoretical Physics* **1958**, *20*, 948–959.

(13) Zwanzig, R. Ensemble Method in the Theory of Irreversibility. *The Journal of Chemical Physics* **1960**, *33*, 1338–1341.

(14) Mori, H. Transport, Collective Motion, and Brownian Motion. *Progress of Theoretical Physics* **1965**, *33*, 423–455.

(15) 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.

(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) Zwanzig, R. Memory Effects in Irreversible Thermodynamics. *Physical Review* **1961**, *124*, 983–992.

(18) Montoya-Castillo, A.; Reichman, D. R. Approximate but Accurate Quantum Dynamics from the Mori Formalism: I. Nonequilibrium Dynamics. *Journal of Chemical Physics* **2016**, *144*, 184104.

(19) Montoya-Castillo, A.; Reichman, D. R. Approximate but Accurate Quantum Dynamics from the Mori Formalism. Ii. Equilibrium Time Correlation Functions. *Journal of Chemical Physics* **2017**, *146*, 084110.

(20) Pfalzgraff, W. C.; Montoya-Castillo, A.; Kelly, A.; Markland, T. E. Efficient construction of generalized master equation memory kernels for multi-state systems from nonadiabatic quantum-classical dynamics. *Journal of Chemical Physics* **2019**, *150*, 244109.

(21) Mulvihill, E.; Schubert, A.; Sun, X.; Dunietz, B. D.; Geva, E. A Modified Approach for Simulating Electronically Nonadiabatic Dynamics Via the Generalized Quantum Master Equation. *Journal of Chemical Physics* **2019**, *150*, 034101.

(22) Mulvihill, E.; Gao, X.; Liu, Y.; Schubert, A.; Dunietz, B. D.; Geva, E. Combining the mapping Hamiltonian linearized semiclassical approach with the generalized quantum master equation to simulate electronically nonadiabatic molecular dynamics. *Journal of Chemical Physics* **2019**, *151*, 074103.

(23) Mulvihill, E.; Lenn, K. M.; Gao, X.; Schubert, A.; Dunietz, B. D.; Geva, E. Simulating energy transfer dynamics in the Fenna-Matthews-Olson complex via the modified generalized quantum master equation. *Journal of Chemical Physics* **2021**, *154*, 204109.

(24) Mulvihill, E.; Geva, E. Simulating the dynamics of electronic observables via reduced-dimensionality generalized quantum master equations. *Journal of Chemical Physics* **2022**, *156*, 044119.

(25) Ng, N.; Limmer, D. T.; Rabani, E. Nonuniqueness of generalized quantum master equations for a single observable. *Journal of Chemical Physics* **2021**, *155*, 156101.

(26) Sayer, T.; Montoya-Castillo, A. Efficient formulation of multitime generalized quantum master equations: Taming the cost of simulating 2D spectra. *Journal of Chemical Physics* **2024**, *160*, 044108.

(27) May, V.; Kühn, O. *Charge and Energy Transfer Dynamics in Molecular Systems*; Wiley-VCH Verlag: Weinheim, 2011.

(28) Lai, Y.; Geva, E. On simulating the dynamics of electronic populations and coherences via quantum master equations based on treating off-diagonal electronic coupling terms as a small perturbation. *Journal of Chemical Physics* **2021**, *155*, 204101.

(29) Sun, X.; Geva, E. Exact vs. asymptotic spectral densities in the Garg-Onuchic-Ambegaokar charge transfer model and its effect on Fermi's golden rule rate constants. *Journal of Chemical Physics* **2016**, *144*, 044106.

(30) Lyu, N.; Mulvihill, E.; Soley, M. B.; Geva, E.; Batista, V. S. Tensor-Train Thermo-Field Memory Kernels for Generalized Quantum Master Equations. *Journal of Chemical Theory and Computation* **2023**, *19*, 1111–1129.

(31) Makarov, D. E.; Makri, N. Path integrals for dissipative systems by tensor multiplication. Condensed phase quantum dynamics for arbitrarily long time. *Chemical Physics Letters* **1994**, *221*, 482.

(32) Wang, H.; Thoss, M. Multilayer Formulation of the Multiconfiguration Time-dependent Hartree Theory. *Journal of Chemical Physics* **2003**, *119*, 1289–1299.

(33) Shi, Q.; Geva, E. A New Approach to Calculating the Memory Kernel of the Generalized Quantum Master Equation for an Arbitrary System-Bath Coupling. *Journal of Chemical Physics* **2003**, *119*, 12063.

(34) Shi, Q.; Chen, L.; Nan, G.; Xu, R.-X.; Yan, Y. Efficient hierarchical Liouville space propagator to quantum dissipative dynamics. *Journal of Chemical Physics* **2009**, *130*, 084105.

(35) Meyer, H.-D., Gatti, F., Worth, G. A., Eds. *Multidimensional Quantum Dynamics: MCTDH Theory and Applications*; Wiley-VCH Verlag: Weinheim, 2009.

(36) Kelly, A.; Brackbill, N.; Markland, T. E. Accurate Nonadiabatic Quantum Dynamics on the Cheap: Making the Most of Mean Field Theory with Master Equations. *Journal of Chemical Physics* **2015**, *142*, 094110.

(37) Shi, Q.; Xu, Y.; Yan, Y.; Xu, M. Efficient propagation of the hierarchical equations of motion using the matrix product state method. *Journal of Chemical Physics* **2018**, *148*, 174102.

(38) Xu, M.; Yan, Y.; Liu, Y.; Shi, Q. Convergence of high order memory kernels in the Nakajima-Zwanzig generalized master equation and rate constants: Case study of the spin-boson model. *Journal of Chemical Physics* **2018**, *148*, 164101.

(39) Dan, X.; Shi, Q. Theoretical study of nonadiabatic hydrogen atom scattering dynamics on metal surfaces using the hierarchical equations of motion method. *Journal of Chemical Physics* **2023**, *159*, 044101.

(40) Lubich, C.; Oseledets, I.; Vandereycken, B. Time Integration of Tensor Trains. *SIAM Journal on Numerical Analysis* **2015**, *53*, 917–941.

(41) Guan, W.; Bao, P.; Peng, J.; Lan, Z.; Shi, Q. mpsqd: A matrix product state based Python package to simulate closed and open system quantum dynamics. *Journal of Chemical Physics* **2024**, *161*, 122501.

(42) Shi, Q.; Geva, E. A Semiclassical Generalized Quantum Master Equation for an Arbitrary System-Bath Coupling. *Journal of Chemical Physics* **2004**, *120*, 10647–10658.

(43) Zhang, M.-L.; Ka, B. J.; Geva, E. Nonequilibrium quantum dynamics in the condensed phase via the generalized quantum master equation. *Journal of Chemical Physics* **2006**, *125*, 044106.

(44) Kelly, A.; Montoya-Castillo, A.; Wang, L.; Markland, T. E. Generalized quantum master equations in and out of equilibrium: When can one win? *Journal of Chemical Physics* **2016**, *144*, 184105.

(45) Levy, E.; Shalit, O. M. Dilation theory in finite dimensions: the possible, the impossible and the unknown. *Rocky Mountain Journal of Mathematics* **2014**, *44*, 203–221.

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

(47) 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.

(48) 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.

Supporting Information for

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

Part VI – The Generalized Quantum Master

Xiaohan Dan,[†,⊥] Pouya Khazaei,[‡,⊥] Brandon Allen,[†] Ningyi Lyu,[†] Callie Wilson,[‡] Ellen Mulvihill,[†]

Yuchen Wang,[¶] Saurabh Shivpuje,[¶] Sabre Kais,[§] Victor S. Batista,[*,†,‖] and Eitan Geva[*,‡]

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

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

[¶]*Department of Chemistry, Purdue University, West Lafayette, IN 47907, USA*

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

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

[⊥]*Contributed equally to this work*

E-mail: victor.batista@yale.edu; eitan@umich.edu

Contents

S.1 Spin–Boson Model

The Spin–Boson Model (SBM) captures a two-level system (donor/acceptor) coupled to a bosonic environment. Below we define the central parameters: diabatic coupling Γ_{DA}, energy bias ϵ, inverse temperature β, the Kondo parameter ξ, and a bath cutoff frequency ω_c. These values set the Hamiltonian and spectral density used throughout.

S.1.1 Model Parameters

The snippet below initializes all physical parameters used by `qflux` modules downstream.

Script S.1.1: Spin–Boson Model parameters

```
GAMMA_DA = 1.0   # diabatic coupling
EPSILON = 1.0    # energy bias
BETA = 5.0      # inverse temperature
XI = 0.1        # Kondo parameter
OMEGA_C = 2.0    # cutoff frequency
```

S.1.2 TT-TFD Reference Dynamics

We employ TT-TFD as a high-fidelity baseline for the reduced density operator (RDO). You can toggle between computing fresh trajectories and loading precomputed data using `Is_run_dynamics`. The plot compares $\sigma_{\mathrm{DD}}(t)$ from TT-TFD with later approximations.

Script S.1.2: Using TT-TFD to simulate the Spin–Boson Model

```
import numpy as np
import matplotlib.pyplot as plt
import qflux
import qflux.GQME.readwrite as wr
import qflux.GQME.params as pa
import matplotlib.pyplot as plt
```

```
# Download the precomputed trajectory data files
# from the qflux GitHub repository https://github.com/batistagroup/qflux
# to the local folder ./GQME_example
from qflux.utils.io import download_github_directory
data_dir = 'GQME_Example_data'
download_github_directory('batistagroup', 'qflux', 'data/GQME_Example', data_dir)

# setup the Hamiltonian and initial state for Spin-Boson Model
Hsys = pa.EPSILON*pa.Z + pa.GAMMA_DA*pa.X
rho0 = np.zeros((pa.DOF_E,pa.DOF_E),dtype=np.complex128)
rho0[0,0] = 1.0

# Create the Spin-Boson model (SBM)
SBM = DynamicsGQME(pa.DOF_E,Hsys,rho0)
SBM.setup_timestep(pa.DT, pa.TIME_STEPS)

# The TT-TFD simulation (may take significant computational time).
# Is_run_dynamics = True  -- Trajectories will be computed and the data will be
    written to `data_dir`
# Is_run_dynamics = False -- Trajectory data will be read from data_dir
Is_run_dynamics = False

if Is_run_dynamics:
   # RDO: reduced density operator, contain the information of population and
    coherence
   # initial_state=0: initial at Donor state
   t, RDO_arr = SBM.tt_tfd(initial_state=0, show_steptime=True, update_type='rk4')
   wr.output_operator_array(t, RDO_arr, f"{data_dir}/TTTFD_Output/TFDSigma_")
else:
   # Read precomputed data from disk
   t, RDO_arr = wr.read_operator_array(pa.TIME_STEPS,
    f"{data_dir}/TTTFD_Output/TFDSigma_")

plt.figure(figsize=(6,4))
plt.plot(t, RDO_arr[:,0].real,'b-', label='TT-TFD')
plt.xlabel(r'$\Gamma t$',fontsize=15)
plt.ylabel(r'$\sigma_{DD}$(t)',fontsize=15)
plt.legend()
plt.show()
```

S.2 Projected Liouvillian

Before introducing memory effects, we construct the system-only projected Liouvillian. This Markovian generator omits bath history and serves as a foil for non-Markovian GQME

behavior.

S.2.1 Constructing the Projected Liouvillian

The following code builds the projected Liouvillian $\langle \mathcal{L} \rangle_n^0$ in the diabatic basis from ϵ and Γ_{DA}.

Script S.2.1: Projected Liouvillian

```
import numpy as np

LN0 = np.zeros((pa.DOF_E_SQ, pa.DOF_E_SQ))
LN0[0][1] = LN0[1][0] = LN0[2][3] = LN0[3][2] = -GAMMA_DA
LN0[0][2] = LN0[2][0] = LN0[1][3] = LN0[3][1] = GAMMA_DA
LN0[1][1] = 2. * EPSILON
LN0[2][2] = -2. * EPSILON
```

S.2.2 Markovian Dynamics and Comparison to TT-TFD

We propagate using $\exp\left(-\mathrm{i}\,\langle \mathcal{L} \rangle_n^0\, t\right)$ and compare $\sigma_{DD}(t)$ to the TT-TFD benchmark, highlighting the role of missing memory.

Script S.2.2: The dynamics with projected Liouvillian only

```
import scipy.linalg as LA

sigma_liou = np.zeros((pa.TIME_STEPS, pa.DOF_E_SQ), dtype=np.complex128)
time_arr = np.linspace(0,(pa.TIME_STEPS-1)*pa.DT,pa.TIME_STEPS)
sigma_liou[0] = np.array([1.0,0,0,0],dtype=np.complex128)
for i in range(1,pa.TIME_STEPS):
  sigma_liou[i] = LA.expm(-1j*LN0*pa.DT)@sigma_liou[i-1]

# read TT-TFD result and plot to compare
timeVec, sigma_tt_tfd =
    wr.read_operator_array(pa.TIME_STEPS,f"{data_dir}/TTTFD_Output/TFDSigma_")
plt.figure(figsize=(6,4))
```

```
plt.plot(time_arr, sigma_liou[:,0].real,'b-', label='Liouvillian only')
plt.plot(timeVec,  sigma_tt_tfd[:,0].real,'ko', markersize=4,markevery=15,
    label='TT-TFD')
plt.xlabel(r'$\Gamma t$',fontsize=15)
plt.ylabel(r'$\sigma_{DD}$(t)',fontsize=15)
plt.legend(loc = (0.22, 0.8))
plt.show()
```

S.3 Memory Kernel for the GQME

The GQME introduces a memory kernel $\mathcal{K}(t)$ that captures non-Markovian bath effects. We now show how to construct $\mathcal{K}(t)$ using **QFlux** and TT-TFD data.

S.3.1 Obtain the Kernel with QFlux

We set up `DynamicsGQME`, prepare H_{sys}, supply (or read) the TT-TFD propagator, and compute the kernel.

Script S.3.1: Using QFlux to obtain the memory kernel

```
from qflux.GQME.dynamics_GQME import DynamicsGQME

# Setup the Hamiltonian and initial state for Spin-Boson Model
Hsys = pa.EPSILON*pa.Z + pa.GAMMA_DA*pa.X
rho0 = np.zeros((pa.DOF_E, pa.DOF_E), dtype=np.complex128)
rho0[0,0] = 1.0

#Create the Spin-Boson model (SBM)
SBM = DynamicsGQME(pa.DOF_E, Hsys, rho0)
SBM.setup_timestep(pa.DT, pa.TIME_STEPS)

# 1. Get the propagator for memory kernel calculation

# The line below calculates all U elements with TT-TFD. The expected waiting time is
    40 minutes on Google Colab.
# To save time, the results are already pre-computed and saved, and Is_run_dynamics
    is therefore set as False.
# The following code would still run normally. Please set Is_run_dynamics = True if
```

```
    one wishes to perform these calculations.

if Is_run_dynamics:
  print('==================now using tt-tfd to calculate propagator')
  timeVec,Gt = SBM.cal_propagator_tttfd()
  print('End of calculate propagator')

  # output the propagator
  wr.output_superoper_array(timeVec, Gt, f"{data_dir}/U_Output/U_")
else:
  timeVec,Gt = wr.read_superoper_array(pa.TIME_STEPS, f"{data_dir}/U_Output/U_")
  SBM.setup_propagator(Gt)

# 2. Volterra scheme: calculating the Memory kernel and output to the file
kernel = SBM.get_memory_kernel()
# output the kernel
wr.output_superoper_array(timeVec, kernel, f"{data_dir}/K_Output/K_")

# Plot the kernel without the last two boundary points that have numerical errors
plt.figure(figsize=(6,4))
plt.plot(timeVec[:-2], kernel[:-2,1,0].real,'b-', label=r'Re
    $\mathcal{K}_{DA,DD}$')
plt.plot(timeVec[:-2], kernel[:-2,0,0].real,'k-', label=r'Re
    $\mathcal{K}_{DD,DD}$')
plt.xlabel(r'$\Gamma t$',fontsize=15)
plt.ylabel(r'$\mathcal{K}$(t)',fontsize=15)
plt.legend(loc = 'upper right')
plt.show()
```

S.3.2 Propagator Superoperator $U(t)$

We compute the full set of initializations needed to reconstruct $U(t)$ for populations and coherences and write the resulting superoperator to disk.

Script S.3.2: The propagator

```
def cal_U_tt_tfd():

   U = np.zeros((pa.TIME_STEPS, pa.DOF_E_SQ, pa.DOF_E_SQ), dtype=np.complex128)

   # tt-tfd with initial state 0,1,2,3
   # initial state |0> means donor state |D>, |3> means acceptor state |A>
```

```
    # |1> is (|D> + |A>)/sqrt(2), |2> is (|D> + i|A>)/sqrt(2)
    print('========calculate the propagator, starting from 0 state========')
    t,U[:,:,0] = tfd.tt_tfd(0)
    print('========calculate the propagator, starting from 1 state========')
    t,U[:,:,1] = tfd.tt_tfd(1)
    print('========calculate the propagator, starting from 2 state========')
    t,U[:,:,2] = tfd.tt_tfd(2)
    print('========calculate the propagator, starting from 3 state========')
    t,U[:,:,3] = tfd.tt_tfd(3)
    print('========calculate the propagator done========')

    U_final = U.copy()

    # the coherence elements that start at initial state |D><A| and |A><D|
    # is the linear combination of above U results
    # |D><A| = |1><1| + i * |2><2| - 1/2 * (1 + i) * (|0><0| + |3><3|)
    U_final[:,:,1] = U[:,:,1] + 1.j * U[:,:,2] - 0.5 * (1. + 1.j) * (U[:,:,0] +
     U[:,:,3])

    # |A><D| = |1><1| - i * |2><2| - 1/2 * (1 - i) * (|0><0| + |3><3|)
    U_final[:,:,2] = U[:,:,1] - 1.j * U[:,:,2] - 0.5 * (1. - 1.j) * (U[:,:,0] +
     U[:,:,3])

    return t,U_final

# The line below calculates all U elements with TT-TFD. The expected waiting
# time is 40 minutes on Google Colab. To save time, the results are already
# pre-computed and saved, and Is_run_dynamics is therefore set as False.
# Set Is_run_dynamics = True to perform these calculations.

if Is_run_dynamics:
    t, Gt = cal_U_tt_tfd()
    # output the propagator
    wr.output_superoper_array(t,Gt,f"{data_dir}/U_Output/U_")
```

S.3.3 Projection-Free Inputs $\mathcal{F}(t)$ and $\dot{\mathcal{F}}(t)$

We differentiate $U(t)$ component-wise to obtain the projection-free inputs that enter the Volterra equation.

Script S.3.3: Projection-Free Inputs $\mathcal{F}(\tau)$ and $\dot{\mathcal{F}}(\tau)$

```
def cal_F():
    #read the propagator data from files
    timeVec,U = wr.read_superoper_array(pa.TIME_STEPS,f"{data_dir}/U_Output/U_")

    F = np.zeros((pa.TIME_STEPS, pa.DOF_E_SQ, pa.DOF_E_SQ), dtype=np.complex128)
    Fdot = np.zeros((pa.TIME_STEPS, pa.DOF_E_SQ, pa.DOF_E_SQ), dtype=np.complex128)

    for j in range(pa.DOF_E_SQ):
        for k in range(pa.DOF_E_SQ):
            # extracts real and imag parts of U element
            Ureal = U[:,j,k].copy().real
            Uimag = U[:,j,k].copy().imag

            # F = i * d/dt U so Re[F] = -1 * d/dt Im[U] and Im[F] = d/dt Re[U]
            Freal = -1. * np.gradient(Uimag.flatten(), pa.DT, edge_order = 2)
            Fimag = np.gradient(Ureal.flatten(), pa.DT, edge_order = 2)

            # Fdot = d/dt F so Re[Fdot] = d/dt Re[F] and Im[Fdot] = d/dt Im[F]
            Fdotreal = np.gradient(Freal, pa.DT)
            Fdotimag = np.gradient(Fimag, pa.DT)

            F[:,j,k] = Freal[:] + 1.j * Fimag[:]
            Fdot[:,j,k] = Fdotreal[:] + 1.j * Fdotimag[:]

    return timeVec,F,Fdot

timeVec,F,Fdot = cal_F()
```

S.3.4 Linear Term $g(t)$

The linear contribution combines $\dot{\mathcal{F}}$ with the Liouvillian action.

Script S.3.4: Linear term $g(t)$

```
linearTerm = 1.j * Fdot.copy() # first term of the linear part
for l in range(pa.TIME_STEPS):
    # subtract second term of linear part
    linearTerm[l,:,:] -= 1./pa.HBAR * F[l,:,:] @ LNO
```

S.3.5 Volterra Solver for $\mathcal{K}(t)$

We iterate a Volterra equation with trapezoidal quadrature until the full kernel converges.

S.3.5.1 Main Iteration

Script S.3.5: Memory Kernel - Volterra Algorithm

```
import time
from tqdm import tqdm

START_TIME = time.time() # starts timing

# sets initial guess to the linear part
prevKernel = linearTerm.copy()
kernel = linearTerm.copy()

# loop for iterations
for numIter in tqdm(range(1, pa.MAX_ITERS + 1)):

    iterStartTime = time.time() # starts timing of iteration
    #print("Iteration:", numIter)

    # calculates kernel using prevKernel and trapezoidal rule
    kernel = CalculateIntegral(F, linearTerm, prevKernel, kernel)

    numConv = 0 # parameter used to check convergence of entire kernel
    for i in range(pa.DOF_E_SQ):
        for j in range(pa.DOF_E_SQ):
            for n in range(pa.TIME_STEPS):
                # if matrix element and time step of kernel is converged, adds 1
                if abs(kernel[n][i][j] - prevKernel[n][i][j]) <= pa.CONVERGENCE_PARAM:
                    numConv += 1

                # if at max iters, prints which elements and time steps did not
                # converge and prevKernel and kernel values
                elif numIter == pa.MAX_ITERS:
                    print("\tK time step and matrix element that didn't converge: %s,
     %s%s"%(n,i,j))

    #print("\tIteration time:", time.time() - iterStartTime)

    # enters if all times steps and matrix elements of kernel converged
    if numConv == pa.TIME_STEPS * pa.DOF_E_SQ * pa.DOF_E_SQ:
        # prints number of iterations and time necessary for convergence
```

```
        print("Number of Iterations:", numIter, "\tVolterra time:", time.time() -
    START_TIME)

        # prints memory kernel to files
        wr.output_superoper_array(timeVec,kernel,f"{data_dir}/K_Output/K_")

        break # exits the iteration loop

    # if not converged, stores kernel as prevKernel, zeros the kernel, and then
    # sets kernel at t = 0 to linear part
    prevKernel = kernel.copy()
    kernel = linearTerm.copy()

    # if max iters reached, prints lack of convergence
    if numIter == pa.MAX_ITERS:
        print("\tERROR: Did not converge for %s iterations"%pa.MAX_ITERS)
        print("\tVolterra time:", print(time.time() - START_TIME))
```

S.3.5.2 Trapezoidal Integral Helper

Script S.3.6: Function to Calculate Integral via Trapezoidal Rule

```
def CalculateIntegral(F, linearTerm, prevKernel, kernel):

    # time step loop starts at 1 because K is equal to linear part at t = 0
    for n in range(1, pa.TIME_STEPS):
        kernel[n,:,:] = 0.

        # f(a) and f(b) terms
        kernel[n,:,:] += 0.5 * pa.DT * F[n,:,:] @ kernel[0,:,:]
        kernel[n,:,:] += 0.5 * pa.DT * F[0,:,:] @ prevKernel[n,:,:]

        # sum of f(a + kh) term
        for c in range(1, n):
            # since a new (supposed-to-be-better) guess for the
            # kernel has been calculated for previous time steps,
            # can use it rather than prevKernel
            kernel[n,:,:] += pa.DT * F[n - c,:,:] @ kernel[c,:,:]

        # multiplies by i and adds the linear part
        kernel[n,:,:] = 1.j * kernel[n,:,:] + linearTerm[n,:,:]

    return kernel
```

S.3.6 Kernel Diagnostics

We visualize representative real parts of kernel elements and omit last boundary points often affected by numerical edge effects.

Script S.3.7: Plot the memory kernel

```
# Plot the kernel without the last two boundary points that have numerical errors
plt.figure(figsize=(6,4))
plt.plot(timeVec[:-2], kernel[:-2,1,0].real,'b-', label=r'Re
    $\mathcal{K}_{DA,DD}$')
plt.plot(timeVec[:-2], kernel[:-2,0,0].real,'k-', label=r'Re
    $\mathcal{K}_{DD,DD}$')
plt.xlabel(r'$\Gamma t$',fontsize=15)
plt.ylabel(r'$\mathcal{K}$(t)',fontsize=15)
plt.legend(loc = 'upper right')
plt.show()
```

S.4 Solving the GQME

With $\mathcal{K}(t)$ in hand, we propagate the reduced density matrix either via a convenience call (`SBM.solve_gqme`) or manually with RK4 to expose the structure of the calculation.

S.4.1 One-Line Solver

Script S.4.1: Solve GQME through QFlux

```
sigma = SBM.solve_gqme(kernel, pa.MEM_TIME)
```

S.4.2 Manual RK4 Propagation

We expose the integrator and the derivative function f used to evolve $\sigma(t)$ under the GQME with finite memory time.

S.4.2.1 RK4 Step

Script S.4.2: GQME - Propagation via RK4 Method

```
def PropagateRK4(currentTime, memTime, kernel,
             sigma_hold, sigma, DT=pa.DT):

   f_0 = Calculatef(currentTime, memTime,
                kernel, sigma, sigma_hold)

   k_1 = sigma_hold + DT * f_0 / 2.
   f_1 = Calculatef(currentTime + DT / 2., memTime,
                kernel, sigma, k_1)

   k_2 = sigma_hold + DT * f_1 /2.
   f_2 = Calculatef(currentTime + DT / 2., memTime,
                kernel, sigma, k_2)

   k_3 = sigma_hold + DT * f_2
   f_3 = Calculatef(currentTime + DT, memTime,
                kernel, sigma, k_3)

   sigma_hold += DT / 6. * (f_0 + 2. * f_1 + 2. * f_2 + f_3)

   return sigma_hold
```

S.4.2.2 Derivative Function

Script S.4.3: Calculating the function f

```
def Calculatef(currentTime, memTime, kernel, sigma_array, kVec, DT=pa.DT,
    HBAR=pa.HBAR, LNO=LNO):

   memTimeSteps = int(memTime / DT)
   currentTimeStep = int(currentTime / DT)

   f_t = np.zeros(kVec.shape, dtype=np.complex128)

   f_t -= 1.j / HBAR * LNO @ kVec

   limit = memTimeSteps
   if currentTimeStep < (memTimeSteps - 1):
      limit = currentTimeStep
   for l in range(limit):
      f_t -= DT * kernel[l,:,:] @ sigma_array[currentTimeStep - l]

   return f_t
```

S.4.2.3 Full Propagation and Benchmarking

We propagate, save $\sigma(t)$, and compare to TT-TFD.

Script S.4.4: GQME - Propagation of the Density Matrix

```
from tqdm import tqdm

# read the memory kernel
timeVec, kernel = wr.read_superoper_array(pa.TIME_STEPS, f"{data_dir}/K_Output/K_")

# array for reduced density matrix elements
sigma = np.zeros((pa.TIME_STEPS, pa.DOF_E_SQ), dtype=np.complex128)

# array to hold copy of sigma
sigma_hold = np.zeros(pa.DOF_E_SQ, dtype = np.complex128)

# sets the initial state at Donor State
sigma[0,0] = 1.
sigma_hold[0] = 1.

# loop to propagate sigma
```

```
print(">>> Starting GQME propagation, memory time =", pa.MEM_TIME)
for l in tqdm(range(pa.TIME_STEPS - 1)): # it propagates to the final time step

    #if l%100==0: print(l)
    currentTime = l * pa.DT
    sigma_hold = PropagateRK4(currentTime, pa.MEM_TIME, kernel, sigma_hold, sigma)
    sigma[l + 1] = sigma_hold.copy()

# prints sigma to files
wr.output_operator_array(timeVec, sigma, f"{data_dir}/GQME_Output/Sigma_")

# Read the reference data and plot
timeVec, sigma_tt_tfd =
    wr.read_operator_array(pa.TIME_STEPS,f"{data_dir}/TTTFD_Output/TFDSigma_")
timeVec, sigma =
    wr.read_operator_array(pa.TIME_STEPS,f"{data_dir}/GQME_Output/Sigma_")

plt.figure(figsize=(6, 4))
plt.plot(timeVec, sigma[:,0].real,'b-', label='GQME')
plt.plot(timeVec, sigma_tt_tfd[:,0].real ,'ko', markersize=4, markevery=15,
    label='benchmark_TT-TFD')
plt.xlabel(r'$\Gamma t$',fontsize=15)
plt.ylabel(r'$\sigma_{DD}$(t)',fontsize=15)
plt.legend()
plt.show()
```

S.5 Quantum Algorithms for GQME via Unitary Dilation

To realize a quantum-circuit-level emulation of non-unitary dynamics, we compute the superpropagator $\mathcal{G}(t)$ and dilate it to a unitary using the Sz.-Nagy construction. The dilated system is then executed on a QASM simulator.

S.5.1 Compute $\mathcal{G}(t)$ from the GQME

Script S.5.1: Calculating $\mathcal{G}(t)$ by solving the GQME

```
from tqdm import tqdm

# read the memory kernel
timeVec,kernel = wr.read_superoper_array(pa.TIME_STEPS, f"{data_dir}/K_Output/K_")

# array for Propagator superoperator elements
G_prop = np.zeros((pa.TIME_STEPS, pa.DOF_E_SQ, pa.DOF_E_SQ), dtype=np.complex128)

# time 0 propagator: identity superoperator
G_prop[0] = np.eye(pa.DOF_E_SQ)

# array to hold copy of G propagator
G_prop_hold = np.eye((pa.DOF_E_SQ), dtype=np.complex128)

# loop to propagate G_prop using GQME
print(">>> Starting GQME propagation, memory time =", pa.MEM_TIME)

for l in tqdm(range(pa.TIME_STEPS - 1)): # it propagates to the final time step

    #if l%100==0: print(l)
    currentTime = l * pa.DT
    G_prop_hold = PropagateRK4(currentTime, pa.MEM_TIME, kernel, G_prop_hold,
     G_prop)
    G_prop[l + 1] = G_prop_hold.copy()
```

S.5.2 Sz.-Nagy Dilation

We map contractions to a larger unitary acting on an extended Hilbert space; this is crucial for implementing non-unitary $\mathcal{G}(t)$ on a gate model.

Script S.5.2: Dilation of the non-unitary propagator

```
from numpy import linalg as la
import scipy.linalg as sp

def dilate(array):

  # Normalization factor of 1.1 to ensure contraction
  norm = LA.norm(array,2)*1.1
  array_new = array/norm
```

```
    ident = np.eye(array.shape[0])

    # Calculate the conjugate transpose of the G propagator
    fcon = (array_new.conjugate()).T

    # Calculate the defect matrix for dilation
    fdef = LA.sqrtm(ident - np.dot(fcon, array_new))

    # Calculate the defect matrix for the conjugate of the G propagator
    fcondef = LA.sqrtm(ident - np.dot(array_new, fcon))

    # Dilate the G propagator to create a unitary operator
    array_dilated = np.block([[array_new, fcondef], [fdef, -fcon]])

    return array_dilated, norm
```

S.5.3 Quantum Simulation via QFlux Interface

We show a direct QFlux route to produce a propagator and set up measurement counts for donor/acceptor populations.

Script S.5.3: Using QFlux to perform quantum algorithm for GQME

```
G_prop = SBM.solve_gqme(kernel, pa.MEM_TIME, dtype='Propagator')

from qflux.open_systems.quantum_simulation import QubitDynamicsOS

qSBM = QubitDynamicsOS(rep='Density', Nsys = pa.DOF_E, Hsys = Hsys, rho0 = rho0)
qSBM.set_count_str(['000','011'])
qSBM.set_dilation_method('Sz-Nagy')

res_qc = qSBM.qc_simulation_vecdens(timeVec, Gprop=G_prop)
pop_qc = res_qc['data']

# visualizing
# Read the exact TT-TFD results
timeVec, sigma_tt_tfd = wr.read_operator_array(pa.TIME_STEPS,
    f"{data_dir}/TTTFD_Output/TFDSigma_")

# Plot the population of the donor and acceptor states
plt.figure(figsize=(6, 4))
```

```
plt.plot(timeVec, pop_qc[:,0], 'r-', label="quantum simulation")
plt.plot(timeVec, sigma_tt_tfd[:,0].real ,'ko', markersize=4, markevery=15,
    label='benchmark_TT-TFD')
plt.xlabel(r'$\Gamma t$',fontsize=15)
plt.ylabel(r'$\sigma_{DD}$(t)',fontsize=15)
plt.legend(loc = 'upper right')
plt.show()
```

S.6 QASM Simulation and Visualization

We now execute the dilated non-unitary dynamics on a QASM backend and reconstruct donor/acceptor populations for comparison to TT-TFD.

S.6.1 Qiskit Dependencies

Script S.6.1: Installing and importing Qiskit dependencies

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, transpile
from qiskit_aer import AerSimulator, QasmSimulator
from qiskit.visualization import plot_histogram
from qiskit.quantum_info import Operator
```

S.6.2 QASM Execution Loop

We iterate over time, apply the dilated propagator, and estimate state populations from shot statistics.

Script S.6.2: QASM simulation for GQME

```
from tqdm import tqdm

# Create a dictionary to store the measurement results
result = {'000': 0, '001': 0, '010': 0, '011': 0, '100': 0, '101': 0, '110': 0,
    '111': 0}

# Create lists to store the population for the acceptor and donor states
pop_accept = []
pop_donor = []

# initial state in the dilated space
rho0_dilated = np.concatenate((np.array([1 + 0j, 0, 0, 0]),np.zeros(pa.DOF_E_SQ)))

pbar = tqdm(range(pa.TIME_STEPS), desc="Running simulation", colour="green",
    leave=True)
for i in pbar:

   qr = QuantumRegister(3)  # Create a quantum register with 3 qubits
   cr = ClassicalRegister(3)  # Create a classical register to store measurement
    results
   qc = QuantumCircuit(qr, cr)  # Combine the quantum and classical registers to
    create the quantum circuit

   # Initialize the quantum circuit with the initial state
   qc.initialize(rho0_dilated, qr)

   # Dilated propagator
   U_G, norm = dilate(G_prop[i])

   # Create a custom unitary operator with the dilated propagator
   U_G_op = Operator(U_G)

   # Apply the unitary operator to the quantum circuit's qubits
   qc.unitary(U_G_op, qr)

   # Measure the qubits and store the results in the classical register
   qc.measure(qr, cr)

   # Run the Simulation and Plot the Results
   simulator = QasmSimulator()
   shots = 2000  # Number of shots
   job = simulator.run(qc,shots=shots)
   counts = job.result().get_counts(qc)

   # Update the result dictionary
   for x in counts:
      result[x] = counts[x]
```

```
    # Calculate the populations of donor and acceptor states from measurement
     probabilities
    pop_d = np.sqrt(result['000'] / shots) * norm  # Multiply by the normalization
     factor
    pop_a = np.sqrt(result['011'] / shots) * norm  # Multiply by the normalization
     factor

    #if i%100==0: print('at',i,'step','population',pop_d,pop_a)

    pop_donor.append(pop_d)  # Stacking the population for the donor state
    pop_accept.append(pop_a)  # Stacking the population for the acceptor state

    pbar.set_postfix(populationD=pop_d, populationA=pop_a)

pbar.close()
```

S.6.3 Visualization and Benchmarking

We compare the QASM-based population estimates to TT-TFD data.

Script S.6.3: Visualizing the Results

```
# Read the exact TT-TFD results
timeVec, sigma_tt_tfd =
    wr.read_operator_array(pa.TIME_STEPS,f"{data_dir}/TTTFD_Output/TFDSigma_")

# Plot the population of the donor and acceptor states
plt.figure(figsize=(6,4))
plt.plot(timeVec, pop_donor, 'r-', label="quantum simulation")
plt.plot(timeVec, sigma_tt_tfd[:,0].real ,'ko', markersize=4, markevery=15,
    label='benchmark_TT-TFD')
plt.xlabel(r'$\Gamma t$',fontsize=15)
plt.ylabel(r'$\sigma_{DD}$(t)',fontsize=15)
plt.legend(loc = 'upper right')
plt.show()
```

S.7 Practical Notes and Tips

- **Performance.** TT-TFD propagator generation can be expensive; keep `Is_run_dynamics=False` to use supplied data during prototyping.
- **Numerics.** Derivatives for $\mathcal{F}(t)$ and $\dot{\mathcal{F}}(t)$ use centered `numpy.gradient`. Ensure `pa.DT` matches the TT-TFD sampling cadence.
- **Stability.** The dilation step normalizes by $1.5\|\mathcal{G}\|_2$ to enforce contraction before Sz.-Nagy dilation.
- **Validation.** Always cross-check $\sigma_{DD}(t)$ from GQME and QASM against TT-TFD to catch discretization or IO issues.