Quandary: Optimal Control for Open Quantum Systems

Stefanie Günther* N. Anders Petersson*

last updated December 16, 2021

Contents

*Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA, USA.

1 Introduction

Quandary numerically simulates and optimizes the time evolution of open quantum systems. The underlying dynamics are modeled by Lindblad's master equation, a linear ordinary differential equation (ODE) describing quantum systems interacting with the environment. Quandary solves this ODE numerically by applying a time-stepping integration scheme and applies a gradient-based optimization scheme to determine optimal control pulses that drive the quantum system to a desired target state. Two optimization objectives are considered: Unitary gate optimization that finds controls to realize a logical quantum gate, and pure-state preparation that aims to drive the quantum system to a desired pure target state, such as the ground state of zero energy level.

Quandary is designed to solve optimal control problems in larger open quantum systems targeting modern high performance computing (HPC) platforms. Quandary utilizes distributed memory computations using the message passing paradigm that enables scalability to large number of compute cores. Implemented in C++, Quandary is portable and its object-oriented implementation allows developers to easily extend the predefined setup to suit their particular simulation and optimization requirements. For example, customized gates for Hamiltonian simulations can easily be added to supplement Quandary's predefined gate set.

This document outlines the mathematical background and underlying equations, and summarizes their implementation in Quandary. For installation instructions and execution, please take a look at the `README.md` file. Further, the configuration file `config_template.cfg` is filled with comments that should help users and developers to set up new simulation and optimization runs, and match the

input options to the equations found in this document. A full documentation is under development. In the meantime, don't hesitate to direct any questions to guenther5@llnl.gov. We also refer to our publications [2, 3].

2 Model equation

Quandary models composite quantum systems consisting of Q subsystems, with n_k energy levels for the k-th subsystem, $k = 0, \dots, Q-1$. The time-evolution of the open quantum system is modeled by Lindblad's master equation:

$$\dot{\rho}(t) = -i(H(t)\rho(t) - \rho(t)H(t)) + \mathcal{L}(\rho(t)) \tag{1}$$

for the density matrix $\rho(t) \in \mathbb{C}^{N\times N}$ with dimension $N := \prod_{k=0}^{Q-1} n_k$, where $H(t)$ denotes the Hamiltonian describing the system and control and $\mathcal{L}(\rho(t))$ denotes the Lindbladian operator that models system-environment interactions as specified below.

The Hamiltonian $H(t)$ is decomposed into a time-independent system part H_d and a time-varying control part $(H_c(t))$ that employs the external control fields: $H(t) = H_d + H_c(t)$. The system Hamiltonian is modeld as

$$H_d := \sum_{k=0}^{Q-1} \left(\omega_k a_k^\dagger a_k - \frac{\xi_k}{2} a_k^\dagger a_k^\dagger a_k a_k + \sum_{l>k} \left(J_{kl} \left(a_k^\dagger a_l + a_k a_l^\dagger \right) - \xi_{kl} a_k^\dagger a_k a_l^\dagger a_l \right) \right) \tag{2}$$

where $\omega_k > 0$ denotes the $0 \to 1$ transition frequency and $\xi_k > 0$ is the self-Kerr coefficient of subsystem k, and the cross resonance coefficients are $J_{kl} > 0$ ("dipole-dipole interaction") and $\xi_{kl} > 0$ ("zz-coupling"). Here, $a_k \in \mathbb{C}^{N\times N}$ denotes the lowering operator acting on subsystem k, which is defined as

$$\begin{aligned} a_0 &:= a^{(n_0)} \otimes I_{n_1} \otimes \cdots \otimes I_{n_{Q-1}} \\ a_1 &:= I_{n_0} \otimes a^{(n_1)} \otimes \cdots \otimes I_{n_{Q-1}} \\ &\vdots \\ a_{Q-1} &:= I_{n_0} \otimes I_{n_1} \otimes \cdots \otimes a^{(n_{Q-1})} \end{aligned} \quad \text{with} \quad a^{(n_k)} := \begin{pmatrix} 0 & 1 & & & \\ & 0 & \sqrt{2} & & \\ & & \ddots & \ddots & \\ & & & & \sqrt{n_k - 1} \\ & & & & 0 \end{pmatrix} \in \mathbb{R}^{n_k \times n_k} \tag{3}$$

where $I_{n_k} \in \mathbb{R}^{n_k \times n_k}$ is the identity matrix.

The action of external control fields on the quantum system is modelled through the control Hamiltonian

$$H_c(t) := \sum_{k=0}^{Q-1} f^k(\vec{\alpha}^k, t) \left(a_k + a_k^\dagger \right) \tag{4}$$

where $f^k(\vec{\alpha}^k, t)$ are real-valued, time-dependent control functions that are parameterized by real-valued parameters $\vec{\alpha}^k \in \mathbb{R}^d$, which are to be determined through optimization.

The Lindbladian operator $\mathcal{L}(\rho(t))$ is assumed to be of the form

$$\mathcal{L}(\rho(t)) = \sum_{k=0}^{Q-1} \sum_{l=1}^{2} \mathcal{L}_{lk} \rho(t) \mathcal{L}_{lk}^\dagger - \frac{1}{2} \left(\mathcal{L}_{lk}^\dagger \mathcal{L}_{lk} \rho(t) + \rho(t) \mathcal{L}_{lk}^\dagger \mathcal{L}_{lk} \right). \tag{5}$$

where the collapse operators $\mathcal{L}_{lk}$ model decay and dephasing processes in the subsystem with $\mathcal{L}_{1k} = \frac{1}{\sqrt{T_1^k}} a_k$ ("T1" – decay) and $\mathcal{L}_{2k} = \frac{1}{\sqrt{T_2^k}} a_k^\dagger a_k$ ("T2" – dephasing) for each subsystem k. The constants $T_l^k > 0$ correspond to the half-life of process l on subsystem k. Typical T1 decay time is between $10 - 100$ microseconds (us). T2 dephasing time is typically about half of T1 decay time.

2.1 Rotational frame approximation

Quandary uses the rotating wave approximation in order to slow down the time scales in the solution of Lindblad's master equations. To that end, the user can specify the rotation frequencies ω_k^r for each oscillator. Under the rotating frame wave approximation, the Hamiltonians are transformed to

$$\begin{aligned}\tilde{H}_d(t) := &\sum_{k=0}^{Q-1} (\omega_k - \omega_k^r)\, a_k^\dagger a_k - \frac{\xi_k}{2} a_k^\dagger a_k^\dagger a_k a_k - \sum_{l>k} \xi_{kl} a_k^\dagger a_k a_l^\dagger a_l \\ &+ \sum_{k=0}^{Q-1} \sum_{l>k} J_{kl} \left(\cos(\eta_{kl} t) \left(a_k^\dagger a_l + a_k a_l^\dagger \right) + i \sin(\eta_{kl} t) \left(a_k^\dagger a_l - a_k a_l^\dagger \right) \right) \qquad (6) \\ \tilde{H}_c(t) := &\sum_{k=0}^{Q-1} \left(p^k(\vec{\alpha}^k, t)(a_k + a_k^\dagger) + i q^k(\vec{\alpha}^k, t)(a_k - a_k^\dagger) \right) \qquad (7)\end{aligned}$$

where $\eta_{kl} := \omega_k^r - \omega_l^r$ are the differences in rotational frequencies between subsystems.

Note that the eigenvalues of the rotating frame Hamiltonian become significantly smaller in magnitude by choosing $\omega_k^r \approx \omega_k$ (so that the first term with $a_k^\dagger a_k$ drops out). This slows down the time variation of the state evolution, hence bigger time-step sizes can be chosen when solving the master equation numerically. We remark that the rotating wave approximation ignores terms in the control Hamiltonian that oscillate with frequencies $\pm 2\omega_k^r$. Below, we drop the tildes on $\tilde{H}_d$ and $\tilde{H}_c$ and use the rotating frame definition of the Hamiltonians to model the system evolution in time.

Using the rotating wave approximatino, the real-valued laboratory frame control functions are written as

$$f^k(\vec{\alpha}^k, t) = 2\mathrm{Re}\left(d^k(\vec{\alpha}^k, t) e^{i\omega_k^r t} \right), \quad d^k(\vec{\alpha}^k, t) = p^k(\vec{\alpha}^k, t) + i q^k(\vec{\alpha}^k, t) \qquad (8)$$

where the rotational frequencies ω_k^r act as carrier waves to the rotating-frame control functions $d^k(\vec{\alpha}^k, t)$.

2.2 Control pulses

The time-dependent rotating-frame control functions $d^k(\vec{\alpha}^k, t)$ are parameterized using N_s piecewise quadratic B-spline basis functions $B_s(t)$ acting as envelope for N_f^k carrier waves:

$$d^k(\vec{\alpha}^k, t) = \sum_{f=1}^{N_f^k} \sum_{s=1}^{N_s} \alpha_{s,f}^k B_s(t) e^{i\Omega_k^f t}, \quad \alpha_{s,f}^k = \alpha_{s,f}^{k(1)} + i\alpha_{s,f}^{k(2)} \in \mathbb{C} \qquad (9)$$

The amplitudes $\alpha_{s,f}^{k(1)}, \alpha_{s,f}^{k(2)} \in \mathbb{R}$ are the control parameters (*design* variables) that Quandary can optimize in order to realize a desired system behavior. In order to ensure that the resulting control pulse starts and ends at zero, Quandary always resets the amplitudes of the first and last two basis functions to zero. Figure 1 shows the B-spline basis functions for $N_s = 10$ with constant amplitudes.

The carrier wave frequencies $\Omega_k^f \in \mathbb{R}$ can be choosen to trigger certain system frequencies. They are provided in the rotating frame, such that the corresponding Lab-frame carrier frequencies become $\omega_k^r + \Omega_k^f$. Those frequencies can be chosen to match the transition frequencies in the lab-frame system Hamiltonian. For example, when $\xi_{kl} << \xi_k$, the transition frequencies satisfy $\omega_k - n\xi_k$. Thus by choosing $\Omega_k^f = \omega_k - \omega_k^r - n\xi_k$, one triggers transition between energy levels n and $n+1$ in subsystem k.

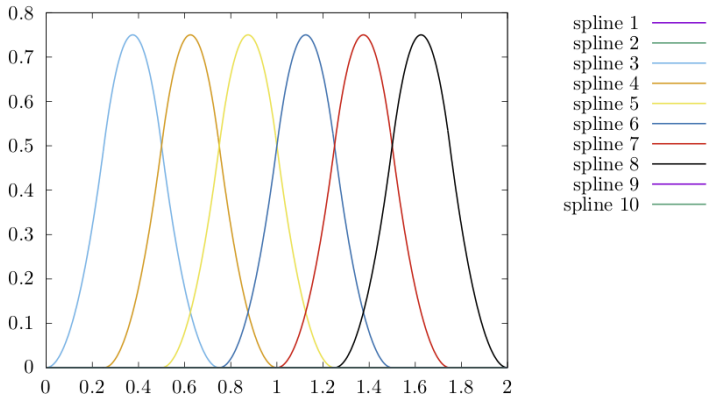

Figure 1: $N_s = 10$ piecewise quadratic B-spline basis functions for constant amplitudes. Note that the first two and last two B-spline amplitudes are always set to zero such that the resulting pulse starts and ends at zero (hence splines 1,2 and 9,10 are not visible in the plot).

Using trigonometric identities, the real and imaginary part of the rotating-frame control $d^k(\vec{\alpha}^k, t) = p^k(\vec{\alpha}^k, t) + iq^k(\vec{\alpha}^k, t)$ are given by

$$p^k(\vec{\alpha}^k, t) = \sum_{f=1}^{N_f^k} \sum_{s=1}^{N_s} B_s(t) \left(\alpha_{s,f}^{k(1)} \cos(\Omega_f^k t) - \alpha_{lsf}^{k(2)} \sin(\Omega_f^k t) \right) \tag{10}$$

$$q^k(\vec{\alpha}^k, t) = \sum_{f=1}^{N_f^k} \sum_{s=1}^{N_s} B_s(t) \left(\alpha_{s,f}^{k(1)} \sin(\Omega_f^k t) + \alpha_{s,f}^{k(2)} \cos(\Omega_f^k t) \right) \tag{11}$$

Those relate to the Lab-frame control $f^k(\vec{\alpha}^k, t)$ through

$$f^k(t) = 2 \sum_{f=1}^{N_f^k} \sum_{s=1}^{N_s} B_s(t) \left(\alpha_{s,f}^{k(1)} \cos((\omega_k^r + \Omega_f^k)t) - \alpha_{s,f}^{k(2)} \sin((\omega_k^r + \Omega_f^k)t) \right) \tag{12}$$

$$= 2p^k(\vec{\alpha}^k, t) \cos(\omega_k^r t) - 2q^k(\vec{\alpha}^k, t) \sin(\omega_k^r t) \tag{13}$$

$$= 2\mathrm{Re}\left(d^k(\vec{\alpha}^k, t) e^{i\omega_k^r t} \right) \tag{14}$$

2.2.1 Storage of the control parameters

The control parameters $\boldsymbol{\alpha}$ are stored in the Quandary code in the following order: List oscillators first $(\vec{\alpha}^0, \dots, \vec{\alpha}^{Q-1})$, then for each $\vec{\alpha}^k \in \mathbb{R}^{2N_s N_f^k}$, iterate over N_s splines: $\vec{\alpha}^k = (\alpha_1^k, \dots, \alpha_{N_s}^k)$ with $\alpha_s^k \in \mathbb{R}^{2N_f^k}$, then each α_s^k iterates over carrier waves and for each carrier wave lists the real and imaginary components: $\alpha_s^k = \alpha_{s,1}^{k(1)}, \alpha_{s,1}^{k(2)}, \dots \alpha_{s,N_f^k}^{k(1)}, \alpha_{s,N_f^k}^{k(2)}$. Hence there are a total of $2N_s \sum_k N_f^k$ real-valued optimization parameters, which are stored in the following order:

$$\boldsymbol{\alpha} := \left(\vec{\alpha}^0, \dots, \vec{\alpha}^{Q-1}\right), \in \mathbb{R}^{2N_s \sum_k N_f^k} \quad \text{where} \tag{15}$$

$$\vec{\alpha}^k = \left(\alpha_{1,1}^k, \dots, \alpha_{1,N_f^k}^k, \dots, \alpha_{N_s,1}^k, \dots, \alpha_{N_s,N_f^k}^k\right), \quad \text{with} \quad \alpha_{s,f}^k = \left(\alpha_{s,f}^{k(1)}, \alpha_{s,f}^{k(2)}\right) \in \mathbb{R}^2, \tag{16}$$

iterating over Q subsystems first, then N_s splines, then N_f^k carrier wave frequencies. To access an element $\alpha_{s,f}^{k(i)}$, $i = 0, 1$, from storage $\boldsymbol{\alpha}$:

$$\alpha_{s,f}^{k(i)} = \boldsymbol{\alpha}[\left(\sum_{j=0}^{k-1} 2N_s N_f^j\right) + s * 2N_f^k + f * 2 + i], \tag{17}$$

When executing Quandary, the control parameter $\boldsymbol{\alpha}$ can be either specified (e.g. a constant pulse, a pi-pulse, or pulses whose parameters are read from a given file), or can be optimized for (Section 3).

In order to guarantee that the optimizer yields control pulses that are bounded with $|p^k(t)| \leq c_{max}^k$, $|q^k(t)| \leq c_{max}^k$ for given bounds c_{max}^k for each subsystem $k = 0, \dots, Q-1$, box constraints are implemented as:

$$|\alpha_{s,f}^{k(1)}| \leq \frac{c_{max}^k}{N_f^k} \quad \text{and} \quad |\alpha_{s,f}^{k(2)}| \leq \frac{c_{max}^k}{N_f^k}. \tag{18}$$

3 The Optimal Control Problem

In the most general form, Quandary can solve the following optimization problem

$$\min_{\boldsymbol{\alpha}} \frac{1}{n_{init}} \sum_{i=1}^{n_{init}} \beta_i J(\rho_i^{target}, \rho_i(T)) + \gamma_1 \int_0^T P\left(\rho_i(t)\right) \, \mathrm{d}t + \gamma_2 \|\boldsymbol{\alpha}\|_2^2 \tag{19}$$

where $\rho_i(T)$ solves Lindblad's master equation (1) in the rotating frame with initial condition $\rho_i(0)$, as specified below. The first term minimizes (a weighted average of) an objective function J that quantifies the discrepancy between a desired target state ρ_i^{target} and the realized state $\rho_i(T)$ at final time T driven by the current control $\boldsymbol{\alpha}$. The second term serves as a penalty that can be added with $\gamma_1 \geq 0$ in oder to achieve a desired behavior of the quantum system over the entire time-domain $0 \leq t \leq T$, see below. The third term is a Tikhonov regularization that can be added with parameter $\gamma_2 \geq 0$ in order to regularize the optimization problem (stabilize optimization convergence) by favoring solutions with small norm.

3.1 Objective function

The following choices for J are implemented

$$J_{Frobenius} = \frac{1}{2}\|\rho^{target} - \rho(T)||_F^2 \tag{20}$$

$$J_{Hilbert-Schmidt} = 1 - \frac{1}{w}\mathrm{Tr}\left((\rho^{target})^\dagger \rho(T)\right), \quad \text{where} \quad w = \mathrm{Tr}(\rho(0)^2) \tag{21}$$

$$J_{Measure} = \mathrm{Tr}\left(N_m \rho(T)\right), \tag{22}$$

where $N_m \in \mathbb{R}^{N\times N}$ is a diagonal matrix with diagonal elements $\lambda_k = |k-m|\,\forall k = 0,\ldots,N-1$ for a given integer m with $0 \leq m < N$. The first two measures are mostly common for gate optimization, $J_{measure}$ is useful when considering pure-state optimization, see below. In Quandary, the objective function can be specified with the configuration option `optim_objective`.

3.2 Optimization targets

Here we describe the target states ρ^{target} that are realized in Quandary. Two cases are considered: State preparation, where the target state is the same for all initial conditions, and gate optimization, where the target state is a unitary transformation of the initial condition. The type of the optimization target can be specified in the configuration option `optim_target`.

3.2.1 Pure target states

State preparation aims to drive the system from either one specific or from any arbitrary initial state to a common desired (fixed) target state. Quandary can optimize towards *pure* target states of the form

$$\rho^{target} = \boldsymbol{e}_m \boldsymbol{e}_m^\dagger, \quad \text{for} \quad m \in \mathbb{N}_0 \quad \text{with} \quad 0 \leq m < N \tag{23}$$

where $\boldsymbol{e}_m$ denotes the m-th unit vector in $\mathbb{R}^N$. We note that considering pure states of that specific form ($\boldsymbol{e}_m \boldsymbol{e}_m^\dagger$) is not a restriction, because any other pure target state can be transformed to this representation using a unitary change of coordinates (compare the Appendix in [3] for a more detailed description). For $m = 0$, the target state represents the ground state of the system under consideration, which has important applications for quantum reset as well as quantum error correction.

Depending on the choice for the initial conditions, optimization towards a pure target state can be used to realize either a simple state-to-state transfer (choosing one specific initial condition, $n_{init} = 1$), or to realize the more complex task of state preparation that drives *any* initial state to a common pure target state, and then $n_{init} > 1$. In the latter case, typically a full basis of initial conditions needs to be considered during the optimization ($n_{init} = N^2$). However, it is shown in [3], that if one chooses the objective function $J_{measure}$ with corresponding measurement operator N_m (see eq. (22)), one can reduce the number of initial conditions to only *one* being an ensemble of all basis states, and hence $n_{init} = 1$ independent of the system dimension N. Compare [3] for details, and Section 3.3.4

Note that the above m refers to the $|m\rangle$-th state of the entire system under consideration with dimension N, which can be a composite of Q subsystems. In Quandary this pure target state is

specified by defining the desired pure target for *each* of the subsystems individually. For a composite system of Q subsystems with n_k levels each, a composite target pure state is specified by a list of integers m_k with $0 \leq m_k < n_k$ representing the pure target state in each subsystem k. The composite pure target is state is then

$$\rho(0) = \Psi\Psi^\dagger, \quad \text{with} \quad \Psi = |m_0 m_1 m_2 \dots m_{Q-1}\rangle \tag{24}$$

$$\text{aka} \quad \Psi = \boldsymbol{e}_{m_0} \otimes \boldsymbol{e}_{m_1} \otimes \cdots \otimes \boldsymbol{e}_{m_{Q-1}} \tag{25}$$

with unit vectors $\boldsymbol{e}_{m_k} \in \mathbb{R}^{n_k}$. The composite-system index m is then computed inside Quandary,from

$$m = m_0 \frac{N}{n_0} + m_1 \frac{N}{n_0 n_1} + m_2 \frac{N}{n_0 n_1 n_2} + \cdots + m_{Q-1} \tag{26}$$

For example, assume one considers a composite system consisting of a qubit modelled with $n_0 = 2$ levels coupled to a readout cavity modelled with $n_1 = 10$ levels, and one wants to drive the qubit to the $|1\rangle$ state and the cavity to the ground state. The target pure state input for Quandary is hence $m_0 = 1, m_1 = 0$ (i.e. the $|10\rangle$ state in the composite system), which corresonds $m = 1 \cdot 10 + 0 = 10$ in the composite system of dimension $N = 2 \cdot 10 = 20$.

3.2.2 Read a target state from file

A specific target state ρ^{target} can also be read from file. This can be useful when considering non-pure target states. File format: The vectorized density matrix (columnwise vectorization), one number per line, first list all real parts, then list all imaginary parts (hence $2N^2$ lines with one number each).

3.2.3 Gate optimization

Quandary can be used to realize logical gate operations. In that case, the target state is not fixed across the initial states, but instead is a unitary transformation of each initial condition. Let $V \in \mathbb{C}^{N \times N}$ be a unitary matrix presenting a logical operation, the goal is to drive any initial state $\rho(0)$ to the unitary transformation $\rho^{target} = V\rho(0)V^\dagger$. Target gates that are currently implemented are

$$V_X := \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad V_Y := \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad V_Z := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad V_{Hadamard} := \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{27}$$

$$V_{CNOT} := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad V_{SWAP} := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{28}$$

Further, a generalization of the SWAP and the CNOT gate for general Q subsystems is available: The SWAP-0Q gate swaps the state of the first and the last qubit, while leaving all other oscillators in their respective initial state, and the CQNOT gate performs a NOT operation on the last qubit if all other qubits are in the one-state. New, user-defined gates can be easily added to the code by extending the `Gate` class in the corresponding `.cpp` and `.hpp` files.

For gate optimization, the first two objective function $J_{Frobenius}$ and $J_{Hilbert-Schmidt}$ are appropriate. Since *any* initial quantum state should be transformed by the control pulses, again typically

a basis for the initial states should be considered ($n_{init} = N^2$). However, it has been shown in [1] that it is enough to optimize with only three specific initial states ($n_{init} = 3$), independent of the Hilbert space dimension N. Those three states are set up in such a way that they can distinguish between any two unitaries in that Hilbert space. The three initial states are readily available in Quandary, see Section 3.3. Note that when optimizing with only those three initial states, it turns out that the choice of the weights β_i that weight the contribution from each initial state in the overall objective function strongly influences the optimization convergence. For faster convergence, it is often benefitial to emphasize on the first of the three initial conditions ($\rho_1(0)$ in Section 3.3.5), hence choosing β_1 (much) bigger than β_2 and β_3 (e.g. $\beta = \{20, 1, 1\}$ often works better than $\beta = \{1, 1, 1\}$, try it yourself). We refer to [1] for details.

3.2.4 Essential and non-essential levels, guard levels

It is often useful, to model the quantum system with more energy levels than the number of levels that the target gate is defined on. For example when optimizing for a SWAP gate on two qubits, with $V_{SWAP} \in \mathbb{C}^{4\times4}$, one might want to model each qubit with more than two energy levels in order to (a) model the (infinite dimensional) system with more accuracy by including more levels and (b) allow the system to transition through higher energy levels in order to achieve the target at final time T. In that case, the *essential* levels denote the levels that the target gate is defined on (or the levels that we actually care about for the optimization target and the initial states.)

To this end, Quandary provides the option to specify the number of *essential* energy levels n_k^e in addition to the number of (total) energy levels n_k, where $n_k^e \leq n_k$ for each subsystem k. The quantum dynamics are then modelled with (more) energy levels with $N = \prod_k n_k$ and $\rho(t) \in \mathbb{C}^{N\times N}$, while the gate is defined in the essential level dimensions only: $V \in C^{N_e\times N_e}, N_e = \prod_k n_k^e$. In the example above, $n_0^e = n_1^e = 2$ and hence $V_{SWAP} \in \mathbb{C}^{4\times4}$, but one can choose the number of energy levels n_0 and n_1 to be bigger than 2 to when modelling the system dynamics.

To compute the objective function at final time T, the essential-dimensional gate is projected upwards to the full dimensions $\tilde{V} \in N \times N$ by inserting identiy blocks for rows/columns that correspond to a non-essential level of either of the subsystems. Hence, a realization of the gate $\tilde{V}$ will not alter the occupation of higher (non-essential) energy level compared to their initial occupation at $t = 0$.

When only the three states as initial conditions are considered (see below), those three initial states will be spanned in the full dimensional system. On the other hand, when the basis of initial states is considered, or their diagonals only, those initial states will be spanned only in the essential level dimensions and zero rows and columns will be inserted for all non-essential levels. Hence, the gate will be realized for any initial state that is spanned in the essential level dimensions, and occupations of non-essential levels at $T = 0$ are avoided.

The occupation of the last non-essential level (the guard-level) per oscillator is penalized throughout the optimization in order to prevent leakage, see section 3.4.

3.3 Initial conditions

Quandary offers various choices to set the initial conditions for simulation and optimization in order to accomodate for different simulation and optimization needs. Below is a list of available options for the initial states (configuration option `initialcondition`).

3.3.1 Pure-state initialization

One can choose to simulate and optimize for only one specific pure initial state (then $n_{init} = 1$). The initial density matrix is then composed of Kronecker products of pure states for each of the subsystems. E.g. for a bipartite system with $n_1 \otimes n_2$ levels, one can propagate any initial pure state

$$\rho(0) = |m_1\rangle\langle m_1| \otimes |m_2\rangle\langle m_2| \quad \text{for} \quad m_1 \in \{0, \dots, n_1 - 1\}, m_2 \in \{0, \dots, n_2 - 1\} \tag{29}$$

Note that, again, in this notation $|m_1\rangle = \boldsymbol{e}_{m_1} \in \mathbb{R}^{n_1}$. The configuration input takes a list of the integers m_k for each subsystem.

3.3.2 Basis of density matrices

To span any possible initial state, an entire basis of states can be used as initial conditions, and then $n_{init} = N^2$. Quandary implements the basis states as defined in [3]:

$$B^{kj} := \frac{1}{2}\left(\boldsymbol{e}_k\boldsymbol{e}_k^\dagger + \boldsymbol{e}_j\boldsymbol{e}_j^\dagger\right) + \begin{cases} 0 & \text{if } k = j \\ \frac{1}{2}\left(\boldsymbol{e}_k\boldsymbol{e}_j^\dagger + \boldsymbol{e}_j\boldsymbol{e}_k^\dagger\right) & \text{if } k < j \\ \frac{i}{2}\left(\boldsymbol{e}_j\boldsymbol{e}_k^\dagger - \boldsymbol{e}_k\boldsymbol{e}_j^\dagger\right) & \text{if } k > j \end{cases} \tag{30}$$

for all $k, j \in \{0, \dots, N-1\}$. These density matrices represent N^2 pure states that span the space of all quantum states in this Hilberspace.

In order to uniquely identify the different initial conditions in the Quandary code and in the output files, a unique index $i \in \{0, \dots, N^2 - 1\}$ is assigned to each basis state with

$$B^i := B^{k(i),j(i)}, \quad \text{with} \quad k(i) := i \bmod N, \quad \text{and} \quad j(i) := \left\lfloor \frac{i}{N} \right\rfloor$$

(columnwise vectorization of a matrix of matrices B^{kj}).

The user can specify a list of integers that identify those subsystems in which the basis should be spanned in. Other oscillators will be initialized with the ground state. To be precise: the user specifies a list of consecutive ID's $< k_0 >, \dots, < k_m >$ with $0 \leq k_j \leq Q - 1$ and $k_{j+1} = k_j + 1$, the basis states B^{kj} will then spanned in the dimension given by those subsystems, $N_s = \prod_{j=0}^{m} n_{k_j}$ and $\rho_s(0) \in \mathbb{C}^{N_s \times N_s}$. The initial states that Quandary propagates are then given by

$$\rho(0) = \boldsymbol{e}_0\boldsymbol{e}_0^\dagger \otimes \underbrace{B^{kj}}_{\in \mathbb{C}^{N_s \times N_s}} \otimes \boldsymbol{e}_0\boldsymbol{e}_0^\dagger \tag{31}$$

where the first $\boldsymbol{e}_0$ (before the kronecker product) is the first unit vector in $\mathbb{R}^{\prod_{k=0}^{k_0-1}}$ (i.e. ground state in all preceding subsystems), and the second $\boldsymbol{e}_0$ (behind the kronecker products) is the first unit vector in the dimension of all subsequent systems, $\mathbb{R}^{\prod_{k=k_m+1}^{Q-1}}$.

Note: The basis matrices are spanned in the dimension defined by the *essential* levels, N_e, if different for the full dimensional system, compare section 3.2.4. They are they lifted up the full-dimension system by inserting rows and columns of zeros at the corresponding positions for each non-essential level.

3.3.3 Diagonal basis matrices (aka all pure states)

One can choose to propagate only those basis elements that correspond to pure states of the form $\boldsymbol{e}_k\boldsymbol{e}_k^\dagger$, i.e. propagating only B^{kk} for $k = 0, \ldots, N-1$, and then $n_{init} = N$.

Similar to the previous section, a list of ID's can be specified in order to identify in which subsystems the diagonal should be spanned, other subsystems will be initialize in the ground state.

Note: The diagonal basis matrices are spanned in the dimension defined by the *essential* levels, N_e, if different for the full dimensional system, compare section 3.2.4. They are they lifted up the full-dimension system by inserting rows and columns of zeros at the corresponding positions for each non-essential level.

3.3.4 Ensemble state for pure-state optimization

For pure-state optimization using the objective function $J_{measure}$ (22), one can use the ensemble state

$$\rho_s(0) = \frac{1}{N^2} \sum_{i,j=0}^{N-1} B^{kj} \tag{32}$$

as the only initial condition for optimization or simulation ($\Rightarrow n_{init} = 1$). Since Lindblad's master equation is linear in the initial condition, and $J_{measure}$ is linear in the final state, propagating this single initial state yields the same target value as if one propagates all basis states spanning that space and averages their measure at final time T (compare [3]).

Similar to the previous two sections, the user can provide a list of integer ID's that determine in which of the subsystems the ensemble state should be spanned. Other subsystems will be initialized in the ground state.

Note: In contrast to the other choices for initial conditions, the ensemble state is currently always spanned in the full dimensional system N (or the full dimensions of the subsystem defined by the IDs), rather than the essential levels. Implementation to operate on essential levels only for the ensemble state is currently missing, sorry!

3.3.5 Three initial states for gate optimization

When considering gate optimization, it has been shown in [1] that it is enough to consider only three specific initial states during optimization ($n_{init} = 3$), independent of the Hilber space dimension. Those three initial states are given by

$$\rho(0)_1 = \sum_{i=0}^{N-1} \frac{2(N-i+1)}{N(N+1)} \boldsymbol{e}_i\boldsymbol{e}_i^\dagger \tag{33}$$

$$\rho(0)_2 = \sum_{ij=0}^{N-1} \frac{1}{N} \boldsymbol{e}_i\boldsymbol{e}_j^\dagger \tag{34}$$

$$\rho(0)_3 = \frac{1}{N} I_N \tag{35}$$

where $I_N \in R^{N\times N}$ is the identity matrix. They are readily implemented in Quandary. Note that it is important to choose the weights $\beta_i, i = 1, 2, 3$ in the objective function appropriately to achieve fast convergence.

Note: The three initial states is spanned in the dimension defined by the *essential* levels, N_e, if different for the full dimensional system, compare section 3.2.4. They are then lifted up the full-dimension system by inserting rows and columns of zeros at the corresponding positions for each non-essential level.

3.3.6 $N + 1$ initial states for gate optimization

The three initial states from above do not suffice to estimate an average fidelity of the realized gate (compare [1]). Instead, it is suggested in that same paper to choose $N + 1$ initial states to compute an average fidelity. Those $N + 1$ initial states consist of the N diagonal states B^{kk} in the Hilbertspace of dimension N, as well as the totally rotated state $\rho(0)_2$ from above. Quandary offers the choice to simulate (or optimize) using those initial states, then $n_{init} = N + 1$.

Note: The $N + 1$ initial states are spanned in the dimension defined by the *essential* levels, N_e, if different for the full dimensional system, compare section 3.2.4. They are then lifted up the full-dimension system by inserting rows and columns of zeros at the corresponding positions for each non-essential level.

3.3.7 Reading an initial state from file

A specific initial state $\rho(0)$ can also be read from file ($\Rightarrow n_{init} = 1$). Format: one column being the vectorized density matrix (vectorization is columnwise), first all real parts, then all imaginary parts (i.e. number of lines is $2N_e^2$, with one value per line).

Note that N_e is the dimension corresponding to the *essential* levels. Ofen, $N_e = N$, but it could be smaller, see Section 3.2.4). If you wish to provide the to-be-read initial state in the *full* dimensions N, you'd have to change that in the source code, inside the constructor of `optimproblem.cpp` (easy).

The option of reading a state from file is useful for example if one wants to propagate a specific *non-pure* initial state. In that case, one first has to generate a datafile storing that state (e.g. by simulating a system and storing and reformating the output), which can then be read in as initial condition.

3.4 Penalty term

The penalty term can be added with parameter $\gamma_1 \geq 0$ in order to achieve a desired behavior at earlier times other than the final time T. In the standard setting, the following penalty term is currently implemented (easily extendable for developers in the `TimeStepper` class.):

$$\gamma_1 \int_0^T P(\rho(t))\mathrm{d}t = \gamma_1 \int_0^T w(t) J\left(\rho(t)\right)\, \mathrm{d}t, \quad \text{where} \quad w(t) = \frac{1}{a} e^{-\left(\frac{t-T}{a}\right)^2}, \tag{36}$$

for a penalty parameter $0 < a \leq 1$. Note, that as $a \to 0$, the weighting function $w(t)$ converges to the Dirac delta distribution with peak at final time T, hence reducing a leads to more emphasis on

the final time T while larger a penalize non-zero energy states at earlier times $t \leq T$. Both γ_1 as well as a can be tuned in the configuration options.

Be aware the adding an integral penalty term requires to evaluate the objective J at *each* time-step within the time-domain $[0, T]$, instead of only the last one. This can be quite expensive, in particular when considering gate optimization, and should hence be turned of if possible (choosing $\gamma_1 = 0$ and/or $a = 0$ will skip the extra computation).

For gate optimization with (non-)essential levels: When the essential levels for gate optimization are smaller than the number of modelled levels ($n_k^e < n_k$ for at least one k), an additional term is added as a penalty in order to prevent leakage into higher levels that are not modelled by the finite-dimensional approximation to the infinite dimensional Hamiltonian. In that case, the occupation of all guard levels (i.e. the last modelled energy levels of each of the subsystems) is penalized:

$$\gamma_1 \int_0^T \sum_r \|\rho(t)_{rr}\|_2^2 \, \mathrm{d}t \tag{37}$$

where $\rho(t)_{rr}$ denotes the diagonal element at (r, r) inside $\rho(t) \in \mathbb{C}^{N \times N}$ and r iterates over all indices that correspond to a guard level (last non-essential energy level) of at least one of the subsystems.

Note that if you want to include this leakage-preventing term, but do *not* want to include the above penalty term (36) (e.g. for efficiency reasons), you can choose $\gamma_1 > 0$ while $a = 0$, which will skip the computation of (36) but will include the term (37).

3.5 Fidelity

As a measure of optimization success, Quandary reports on the fidelity computed from

$$F = \frac{1}{n_{init}} \sum_{i=1}^{n_{init}} \mathrm{Tr}\left(\left(\rho_i^{target} \right)^\dagger \rho_i(T) \right). \tag{38}$$

Note that this fidelity is averaged over the chosen initial conditions, so the user should be careful how to interpret this number. E.g. if one optimizes for a logical gate while choosing the three initial condition as in Section 3.3.5, the fidelity that is reported during optimization will be averaged over those three initial states, which is not sufficient to estimate the actual average fidelity over the entire space of potential intial states. It is advised to recompute the average fidelity **after** optimization has finished, e.g. by propagating the N^2 basis states B_{kj} to final time T using the optimized control parameter, or by propagating only $N + 1$ initial states to get an estimate thereof.

Note that for pure-state preparation, we have $\rho^{target} = \boldsymbol{e}_m \boldsymbol{e}_m^\dagger$ and therefor the fidelity can be computed from $F = \frac{1}{n_{init}} \sum_i \left[\rho_i(T)\right]_{mm}$, for the m-th diagonal element of the final states $\rho_i(T)$.

3.6 Multi-level initialization: B-spline refinement

Quandary offers the possibility to progressively refine the B-spline basis functions between optimization runs. This is a useful feature that can tackle the overparameterization problem by starting the optimizer with very few B-splines per oscillator (e.g. with the minimum $N_s = 5$), and successively refine the number of basis functions when the optimizer stalls out or stops at a local minimum. During refinement, each B-spline basis function splits into 4 children, compare Figure 2.

Note neighbouring mother B-splines share two of the child splines since they overlap, and two child splines will be entirely outside of [0,T]. Therefore, the number of splines N_s will refine into $2(N_s-1)$ B-spline basis functions during refinement. Due to the hierarchical structure of the B-splines, the refined splines with corresponding refined parameters will produce the exact same control pulse as the non-refined splines. However it provides more degrees of freedom for the optimizer (more control parameters), such that further optimization on the refined parameters (hopefully) results in a better minimum than the non-refined paramter.

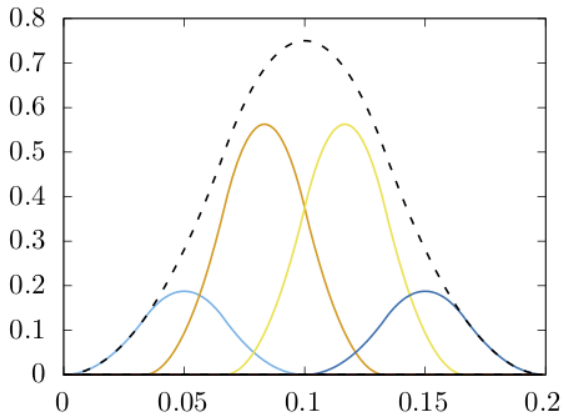

Figure 2: B-spline refinement: Each B-spline basis function (dashed) splits into 4 children wavelets (colors), with weights $[\frac{1}{4}, \frac{3}{4}, \frac{3}{4}, \frac{1}{4}]$. The number of total splines during refinement increases from N_s to $2(N_s-2)$ due to overlapping child splines 2 child splines being outside of [0,T].

Refinement can be enabled through the configuration option `optim_init` (take a look at the `config_template.cfg` file). If enabled, Quandary will read in the provided parameter file, and will refine each B-spline basis function before starting the optimization run (refinement is happening during the optimizer initialization phase, when setting the initial guess for the optimization parameters). Pay attention to the screen output log to see the refined number of splines and number of optimizaiton paramters.

Be careful when setting up the configuration for refinement: Make sure that the configuration option `nspline` reflects the number of splines that had been used to generate the provided parameter file that is beeing red in (i.e. provide the number N_s, *pre*-refinement, rather than the number $2(N_s-1)$ that are to used *post*- refinement). You you verify, that the refined splines and refined paramter file results in the same control pulses that were generated from the non-refined splines. (If it doesn't, it's likely that the configuration options are not set up correctly: did you use the correct number of splines, number of carrierwaves, correct parameter file? Reach out if you can't get it to work.)

4 Implementation

4.1 Vectorization

Quandary solves Lindblad's master equation (1) in vectorized form with $q(t) := \mathrm{vec}(\rho(t)) \in \mathbb{C}^{N^2}$ (columnwise vectorization). Using the relations

$$\mathrm{vec}(AB) = (I_N \otimes A)\mathrm{vec}(B) = (B^T \otimes I_N)\mathrm{vec}(A) \tag{39}$$

$$\mathrm{vec}(ABC) = (C^T \otimes A)\mathrm{vec}(B) \tag{40}$$

for square matrices $A, B, C \in \mathbb{C}^{N\times N}$, the vectorized form of the Lindblad master equation is given by:

$$\dot{q}(t) = M(t)q(t) \quad \text{where} \tag{41}$$

$$M(t) := -i(I_N \otimes H(t) - H(t)^T \otimes I_N) + \sum_{k=0}^{Q-1}\sum_{l=1}^{2} \gamma_{lk}\left(\mathcal{L}_{lk} \otimes \mathcal{L}_{lk} - \frac{1}{2}\left(I_N \otimes \mathcal{L}_{lk}^T\mathcal{L}_{lk} + \mathcal{L}_{lk}^T\mathcal{L}_{lk} \otimes I_N\right)\right) \tag{42}$$

with $M(t) \in \mathbb{C}^{N^2\times N^2}$, and $H(t) = H_d(t) + H_c(t)$ being the rotating frame system and control Hamiltonians as in (6) and (7), respectively.

4.1.1 Real-valued system and state storage

Quandary solves the vectorized master equation (41) in real-valued variables with $q(t) = u(t)+iv(t)$, evolving the real-valued states $u(t), v(t) \in \mathbb{R}^{N^2}$ with

$$\dot{q}(t) = M(t)q(t) \quad \Leftrightarrow \quad \begin{bmatrix}\dot{u}(t)\\ \dot{v}(t)\end{bmatrix} = \begin{bmatrix}A(t) & -B(t)\\ B(t) & A(t)\end{bmatrix}\begin{pmatrix}u(t)\\ v(t)\end{pmatrix} \tag{43}$$

for real and imaginary parts $A(t) = \mathrm{Re}\,(M(t))$ and $B(t) = \mathrm{Im}\,(M(t))$.

The real and imaginary parts of $q(t)$ are stored in a colocated manner: For $q = u + iv$ with $u, v \in \mathbb{R}^{N^2}$, a vector of size $2N^2$ is stored that staggers real and imaginary parts behind each other for each component:

$$q = u + iv = \begin{bmatrix}u^1\\ u^2\\ \vdots\\ u^{N^2-1}\end{bmatrix} + i\begin{bmatrix}v^1\\ v^2\\ \vdots\\ v^{N^2-1}\end{bmatrix} \quad \Rightarrow \quad q_{store} = \begin{bmatrix}u_1\\ v_1\\ u_2\\ v_2\\ \vdots\\ u_{N^2-1}\\ v_{N^2-1}\end{bmatrix}$$

4.2 Sparse-matrix vs. matrix-free solver

In Quandary, two versions to evaluate the right hand side of Lindblad's equation, $M(t)q(t)$, of the vectorized real-valued system are available:

- The *sparse-matrix solver* uses PETSc's sparse matrix format (sparse AIJ) to set up (and store) the time-independent building blocks inside $A(t)$ and $B(t)$. Sparse matrix-vector products are then applied at each time-step to evaluate the products $A(t)u(t) - B(t)v(t)$ and $B(t)u(t) + A(t)v(t)$.

  For developers, the appendix provides details on each term within $A(t)$ and $B(t)$ which can be matched to the implementation in the code (class `MasterEq`).

- The *matrix-free solver* considers the state density matrix $\rho \in C^{N\times N}$ to be a tensor of rank $2Q$ (one axis for each subsystems for each matrix dimension, hence $2 \cdot Q$ axes). Instead of storing the matrices within $M(t)$, the matrix-free solver applies tensor contractions to realize the action of $A(t)$ and $B(t)$ on the state vectors.

  In our current test cases, the matrix-free solver is much faster than the sparse-matrix solver (about 10x), no surprise. However the matrix-free solver is currently only implemented for systems consisting of **2, 3, 4, or 5** subsystems.

  Important note: The matrix-free solver currently does not parallelize across the system dimension N, hence the state vector is **not** distributed (i.e. no parallel Petsc!). The reason why we did not implement that yet is that Q can often be large while each axis can be very shor (e.g. modelling $Q = 12$ qubits with $n_k = 2$ energy levels per qubit), which yields a very high-dimensional tensor with very short axix. In that case, the standard (?) approach of parallelizing the tensor along its axes will likey lead to very poor scalability due to high communication overhead. We have not found a satisfying solution yet - if you have ideas, please reach out, we are happy to collaborate!

4.3 Time-stepping

To solve the vectorized master equation (41), $\dot{q}(t) = M(t)q(t)$ for $t \in [0, T]$, Quandary applies a time-stepping integration scheme on a uniform time discretization grid $0 = t_0 < \dots t_N = T$, with $t_n = n\delta t$ and $\delta t = \frac{T}{N}$, and approximates the solution at each discrete time step $q^n \approx q(t_n)$. It is a second-order accurate, symplectic Runge-Kutta scheme tableau $\begin{array}{c|c} 1/2 & 1/2 \\ \hline & 1 \end{array}$. Given a state q^n at time t_n, the update formula to compute q^{n+1} is hence

$$q^{n+1} = q^n + \delta t k_1 \quad \text{where } k_1 \text{ solves} \quad \left(I - \frac{\delta t}{2} M^{n+1/2}\right) k_1 = M^{n+1/2} q^n \tag{44}$$

where $M^{n+1/2} := M(t_n + \frac{\delta t}{2})$. In each time-step, we first solve a linear equation to get the stage variable k_1, then use it to update q^{n+1}.

While the implicit midpoint rule is the default (and preferred) setting in Quandary, there is the possibility to apply any time-stepping integrators provided by PETSc (class `TS` within PETSc). However, implementation needs to be verified, and adjoint time-stepping with Petsc is currently not available.

4.3.1 Choice of the time-step size

In order to choose a time-step size δt, an eigenvalue analysis of the constant drift Hamiltonian H_d is often useful. Since H_d is Hermitian, there exists a transformation Y such that $Y^\dagger H_d Y = \Lambda$ where $Y^\dagger = Y$ where Λ is a diagonal matrix containing the eigenvalues of H_d. Transform the state $\tilde{q} = Y^\dagger q$, then the ODE transforms to

$$\dot{\tilde{q}} = -i\Lambda\tilde{q} \quad \Rightarrow \dot{\tilde{q}}_i = -i\lambda_i\tilde{q}_i \quad \Rightarrow \tilde{q}_i = a\exp(-i\lambda_i t)$$

Therefore, the period for each mode is $\tau_i = \frac{2\pi}{|\lambda_i|}$, hence the shortest period is $\tau_{min} = \frac{2\pi}{\max_i\{|\lambda_i|\}}$. If we want p discrete time points per period, then $p\delta t = \tau_{min}$, hence

$$\delta t = \frac{\tau_{min}}{p} = \frac{2\pi}{p\max_i\{|\lambda_i|\}} \tag{45}$$

Usually, for a first order scheme we would use something like $p = 20$, second order may use $p = 10$. The above estimate provides a first idea on how big (small) the time-step size should be, and the user is advised to consider this estimate himself when running a test case. However, the estimate ignores contributions from the control Hamiltonian, where larger control amplitudes will require smaller and smaller time-steps in order to resolve (a) the time-varying controls themselves and (b) the dynamics induced by large control contributions. A standard Δt test should be performed to varify if the time-step is small enough (testing for second order convergence of the error when reducing Δt).

If one wants to include the time-varying Hamiltonian part $H = H_d + H_c(t)$ in the analysis, one could use the constraints on the control parameter amplitudes to remove the time-dependency using their large value instead.

4.4 Gradient computation via discrete adjoint backpropagation

Quandary computes the gradients of the objective function with respect to the design variables $\boldsymbol{\alpha}$ using the discrete adjoint method. The discrete adjoint approach yields exact and consistent gradients on the algorithmic level, at costs that are independent of the number of design variables. To that end, the adjoint approach propagates local sensitivities backwards through the time-domain while concatenating contributions to the gradient using the chain-rule.

The consistent discrete adjoint time-integration step for adjoint variables denoted by $\bar{q}^n$ is given by

$$\bar{q}^n = \bar{q}^{n+1} + \delta t\left(M^{n+1/2}\right)^T\bar{k}_1 \quad \text{where}\,\bar{k}_1\,\text{solves} \quad \left(I - \frac{\delta t}{2}M^{n+1/2}\right)^T\bar{k}_1 = \bar{q}^{n+1} \tag{46}$$

The contribution to the gradient ∇J for each time step is

$$\nabla J+ = \delta t\left(\frac{\partial M^{n+1/2}}{\partial z}\left(q^n + \frac{\delta t}{2}k_1\right)\right)^T\bar{k}_1 \tag{47}$$

Each evaluation of the gradient ∇J involes a forward solve of n_{init} initial quantum states to evaluate the objective function at final time T, as well as n_{init} backward solves to compute the adjoint states and the contributions to the gradient. Note that the gradient computation (47) requires the states and adjoint states at each time step, hence the states q^n need to be stored during forward propagation.

4.5 Optimization algorithm

Quandary utilized Petsc's `Tao` optimization package to apply gradient-based iterative updates to the control variables. The `Tao` optimization interface takes routines to evaluate the objective function as well as the gradient computation. In the current setting in Quandary, `Tao` applies a nonlinear Quasi-Newton optimization scheme using a preconditioned gradient based on L-BFGS updates to approximate the Hessian of the objective function. A projected line-search is applied to ensure that the objective function yields sufficient decrease per optimization iteration while keeping the control parameters within the prescribed box-constraints.

4.6 Parallelization

Quandary offers three levels of parallelization using MPI.

1. Parallelization over initial conditions: The n_{init} initial conditions $\rho_i(0)$ can be distributed over `np_init` compute units. Since initial condition are propagated throught the Lindblad solver independently from each other, speedup from distributed initial conditions is ideal.
2. Parallel linear algebra with Petsc (sparse-matrix solver only): For the sparse-matrix solver, Quandary utilizes Petsc's parallel sparse matrix and vector storage to distribute vectors onto `np_petsc` compute units. To perform scaling results, make sure to disable code output (or reduce the output frequency to print only the last time-step), because writing the data files invokes additional MPI calls to gather data on the master node.
3. Time-parallelization via XBraid (experts only): working implementation, but speedup depends on the test case and XBraid setting (experts only), configuration option is `np_braid`

Strong scaling studies are presented in [3].

In the main code, the global communicator (MPI_COMM_WORLD) is split into three sub-communicator, one for each of the above. The total number of executing MPI processes (np_{total}) is split into three subgroups in such a way that

$$np_{braid} * np_{init} * np_{petsc} = np_{total}.$$

The user specifies the size of the communicator for distributing the initial conditions (np_{init}) as well as time-parallelization (np_{braid}) in the configuration file. The number of cores for parallel linear algebra (np_{petsc}) is then computed from the above equation. The following requirements for parallel distribution must be considered when setting up parallel runs:

- $\frac{n_{init}}{np_{init}} \in \mathbb{N}$, where n_{init} is the number of initial conditions that are considered (being N^2 for the full basis, N for considering diagonals only as initial condition, and 1 if propagating only one initial condition e.g. reading from file, or pure state initialization). This requirement ensures that each processor group owns the same number of initial conditions.
- $\frac{np_{total}}{np_{init} * np_{braid}} \in \mathbb{N}$, so that each processor group has the same number of cores for Petsc.
- $\frac{N^2}{np_{petsc}} \in \mathbb{N}$, hence the system dimensions must be integer multiple of the number of cores for distributing linear algebra in Petsc. This constraint is a little annoying, however the current implementation requires this due to the colocated storage of the real and imaginary parts of the vectorized state.

4.7 Testing

- Quandary has a set of regression tests. Please take a look at the REGRESSIONTEST.md document in the `quandary/tests` directory for instruction on how to run the regression tests.
- In order to check if the gradient implementation is correct, one can choose to run a Central Finite Difference test. Let the overall objective function be denoted by $F(\boldsymbol{\alpha})$. The Central Finite Difference test compares each element of the gradient $\nabla F(\boldsymbol{\alpha})$ with the following (second-order accurate) estimate:

$$\left(\nabla F(\boldsymbol{\alpha})\right)_i \approx \frac{F(\boldsymbol{\alpha} + \epsilon \boldsymbol{e}_i) - F(\boldsymbol{\alpha} - \epsilon \boldsymbol{e}_i)}{2\epsilon} \qquad \text{(CFD)}$$

  for unit vectors $\boldsymbol{e}_i \in \mathbb{R}^d$, and d being the dimension of $\boldsymbol{\alpha}$.

  To enable the test, set the flag for the compiler directive `TEST_FD_GRAD` at the beginning of the `src/main.cpp` file. Quandary will then iterate over all elements in α and report the *relative* error of the implemented gradient with respect to the "true" gradient computed from CFD.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-SM-818073.

References

[1] Michael H Goerz, Daniel M Reich, and Christiane P Koch. Optimal control theory for a unitary operation under dissipative evolution. *New Journal of Physics*, 16(5):055012, 2014.

[2] Stefanie Günther, N. Anders Petersson, and Jonathan L. DuBois. Quandary: An open-source c++ package for high-performance optimal control of open quantum systems. https://arxiv.org/abs/2110.10310, 2021.

[3] Stefanie Günther, N. Anders Petersson, and Jonathan L. DuBois. Quantum optimal control for pure-state preparation using one initial state. *AVS Quantum Science*, 3:043801, 2021.

A Appendix: Details for the real-valued, vectorized Hamiltonian

To assemble (evaluate) $A(t) = Re(M(t))$ and $B(t) = Im(M(t))$, consider

$$iH = iH_d(t) + iH_c(t) \tag{48}$$

$$= i\left(\sum_k(\omega_k - \omega_k^{\text{rot}})a_k^\dagger a_k - \frac{\xi}{2}a_k^\dagger a_k^\dagger a_k a_k - \sum_{l>k}\xi_{kl}a_k^\dagger a_k a_l^\dagger a_l + \sum_{l>k}J_{kl}\cos(\eta_{kl}t)\left(a_k^\dagger a_l + a_k a_l^\dagger\right)\right. \tag{49}$$

$$\left. + \sum_k p^k(\vec{\alpha}^k, t)\left(a_k + a_k^\dagger\right)\right) \tag{50}$$

$$+\left(\sum_k\sum_{kl} -J_{kl}\sin(\eta_{kl}t)\left(a_k^\dagger a_l - a_k a_l^\dagger\right) - \sum_k q^k(\vec{\alpha}^k, t)\left(a_k - a_k^\dagger\right)\right) \tag{51}$$

Hence $A(t)$ and $B(t)$ are given by

$$A(t) = A_d + \sum_k q^k(\vec{\alpha}^k, t)A_c^k + \sum_{l>k}J_{kl}\sin(\eta_{kl}t)A_d^{kl} \tag{52}$$

$$\text{with} \quad A_d := \sum_k\sum_{j=1,2}\gamma_{jk}\left(\mathcal{L}_{jk}\otimes\mathcal{L}_{jk} - \frac{1}{2}\left(I_N\otimes\mathcal{L}_{jk}^T\mathcal{L}_{jk} + \mathcal{L}_{jk}^T\mathcal{L}_{jk}\otimes I_N\right)\right) \tag{53}$$

$$A_c^k := I_N\otimes\left(a_k - a_k^\dagger\right) - \left(a_k - a_k^\dagger\right)^T\otimes I_N \tag{54}$$

$$A_d^{kl} := I_N\otimes\left(a_k^\dagger a_l - a_k a_l^\dagger\right) - \left(a_k^\dagger a_l - a_k a_l^\dagger\right)^T\otimes I_N \tag{55}$$

and

$$B(t) = B_d + \sum_k p^k(\vec{\alpha}^k, t)B_c^k + \sum_{kl}J_{kl}\cos(\eta_{kl}t)B_d^{kl} \tag{56}$$

$$\text{with} \quad B_d := \sum_k(\omega_k - \omega_k^{\text{rot}})\left(-I_N\otimes a_k^\dagger a_k + (a_k^\dagger a_k)^T\otimes I_N\right) - \frac{\xi_k}{2}\left(-I_N\otimes a_k^\dagger a_k^\dagger a_k a_k + (a_k^\dagger a_k^\dagger a_k a_k)^T\otimes I_N\right) \tag{57}$$

$$- \sum_{l>k}\xi_{kl}\left(-I_N\otimes a_k^\dagger a_k a_l^\dagger a_l + (a_k^\dagger a_k a_l^\dagger a_l)^T\otimes I_N\right) \tag{58}$$

$$B_c^k := -I_N\otimes\left(a_k + a_k^\dagger\right) + \left(a_k + a_k^\dagger\right)^T\otimes I_N \tag{59}$$

$$B_d^{kl} := -I_N\otimes\left(a_k^\dagger a_l + a_k a_l^\dagger\right) + \left(a_k^\dagger a_l + a_k a_l^\dagger\right)^T\otimes I_N \tag{60}$$

$$\tag{61}$$

The sparse-matrix solver initializes and stores the constant matrices $A_d, A_d^{kl}, A_c^k, B_d, B_d^{kl}, B_c^k$ using Petsc's sparse-matrix format. They are used as building blocks to evaluate the blocks in the system matrix $M(t)$ with

$$A(t) = Re(M(t)) = A_d + \sum_k q^k(\alpha^k, t)A_c^k + \sum_{l>k}J_{kl}\sin(\eta_{kl}t)A_d^{kl} \tag{62}$$

$$B(t) = Im(M(t)) = B_d + \sum_k p^k(\alpha^k, t)B_c^k + \sum_{kl}J_{kl}\cos(\eta_{kl}t)B_d^{kl} \tag{63}$$

at each time t, which are applied to the vectorized, real-valued density matrix using Petsc's sparse MatVec implementation.

The matrix-free solver does not explicitly store the matrices A_d, B_d, A_c^k, B_c^k, etc., but instead only evaluates their action on a vector $q(t)$ using tensor contractions applied to the corresponding dimension of the density matrix tensor.