

大規模量子系の最適制御：GRAPEのメモリ性能とランタイム性能の評価

Yunwei Lu,^{1,*} Sandeep Joshi,¹ Vinh San Dinh,¹ and Jens Koch¹.

ノースウェスタン大学物理学・天文学科、イリノイ州エバンストン、60208、米国（日付：2023年4月14日）

勾配上昇パルス工学（GRAPE）は量子最適制御でよく使われる手法で、自動微分（AD）と組み合わせることで、コスト関数の勾配をその場で評価することが可能になる。しかし、ADの利便性は、多数の状態やプロパゲータを累積的に保存することによるメモリコストと引き替えになることを説明する。ヒルベルト空間のサイズが大きくなる量子系では、これが大きなボトルネックとなる。我々は、勾配をハードコードする戦略を再検討し、プロパゲータの保存を完全に回避し、メモリ要件を大幅に削減する方式を提案する。これとは別に、実行時性能を向上させるための数値状態伝播の改良を発表した。ランタイムとメモリ使用量のベンチマークを行い、より大きなヒルベルト空間サイズへの推進に焦点を当て、このアプローチをADベースの実装と比較する。その結果、ADを用いないアプローチは、他の方法では取り組むことが困難な大規模量子系への最適制御の適用を容易にすることが確認された。

I. INTRODUCTION

急速に発展する量子情報処理分野では、状態準備[1]、エラー訂正[2, 3]、論理ゲートの実現[4-7]などのタスクに対する量子最適制御が大きな関心を呼んでいる。この手法は、核磁気共鳴[9-14]、トラップイオン[15, 16]、ダイヤモンドの窒素空孔中心[17-23]、ポーズ-AINシュタイン凝縮[24-27]、中性原子[28, 29]など様々な量子系[8]で実施されてきた。量子最適制御を実現する著名な手法の一つに、勾配上昇パルス工学（GRAPE）があります。最適制御の中心的な目標は、所望の状態遷移やゲート動作の忠実度を最小化するように制御パラメータを調整することである。GRAPEでは、最小化は勾配降下に基づくため、一般に勾配を評価する必要がある。自動微分法（AD）[30]は、GRAPEを用いた量子最適制御にも影響を与えており便利なツールである[31-33]。ADは内部的には計算グラフを構築し、連鎖法則を適用して与えられた関数の勾配を自動計算する。しかし、ADの利便性は、完全な計算グラフを保存するために必要なメモリという代償を払うことになる。半自動微分法（semi-AD）[34]を用いると、完全自動微分法（full-AD）に比べてメモリのオーバーヘッドを部分的に削減することができます。しかし、この削減は、サイズが急激に大きくなる量子系の最適制御に取り組むにはまだ不十分である[35, 36]。そこで、ハードコードされた勾配（HG）に基づき、最小のメモリコストで実現できるオリジナルのGRAPEスキーム[11]を参考することにしました。メモリ使用量と実行時間のスケーリング分析を行い、HG、semi-AD、full-ADの中から最適な戦略を選択するための決定木を開発した。具体的な最適化タスクのベンチマークを用いて、スケーリング動作を例証する。本論文のアウトラインは以下の通りである。セクションIIでは、GRAPEを簡単に概説し、必要な表記法を紹介する。セクションIIIでは、最もよく使われるコスト関数に対する勾配の導出とメモリ効率の良い実装を説明する。

セクションIVでは、HGの数値実装により、状態伝播の効率はさらに向上する。Sec. IVでは、HG、AD、semi-ADの実行時間とメモリ使用量のスケーリングを分析・比較し、具体的なベンチマーク研究の結果を示す。Sec. Vでは、GRAPEに基づく量子最適制御の最適な数値戦略の選択を容易にする決定木を定式化する。また、Sec. VIでは、これまでの研究成果をまとめ、結論を示す。

II. BRIEF SKETCH OF GRAPE

GRAPE[11]の基本を簡単にスケッチし、主に以降のセクションで使用する概念を確立する。以下のハミルトニアンによって記述される系を考える。

$$H(t) = H_s + a(t) h_c. \quad (1)$$

ここで、 H_s は静的システムのハミルトニアン、 h_c は古典制御場 $a(t)$ をシステム¹に結合する演算子である。量子最適制御は、与えられた時間間隔 $0 \leq t \leq T$ の間に、所望のユニタリーゲートまたは状態遷移が行われるように $a(t)$ を調整することを目的としている。最適化を容易にするため、全制御時間 T を N 個の時間間隔 $\Delta t = T / N$ に分割し、離散時間 $t_n = n\Delta t$ ($n = 1, 2, \dots, N$)における振幅で制御場を規定する。離散化間隔 Δt は、各 Δt 内で制御振幅がほぼ一定²となるように選ばれる。時間離散化した値を次のように表す。

$$a_n := a(t_n), \quad (2)$$

とし、これらをグループ化してベクトル $a \in \mathbb{R}^N$ を形成する。この区別的定数制御場の影響を受けて、システムの

¹ 簡単のため、実数値の $a(t)$ を持つ1つの制御チャネルを考えることにする。複数の制御チャネルと複雑な a に対する一般化は簡単です。この扱いは、実際には、任意の波形発生器の実際の出力をモデル化するのに非常に近く、 t は、利用可能な分解能に

² 合わせて選択することができます。

* yunweilu2020@u.northwestern.edu

Optimal control of large quantum systems: assessing memory and runtime performance of GRAPE

Yunwei Lu,^{1,*} Sandeep Joshi,¹ Vinh San Dinh,¹ and Jens Koch¹

¹*Department of Physics and Astronomy, Northwestern University, Evanston, Illinois 60208, USA*

(Dated: April 14, 2023)

Gradient Ascent Pulse Engineering (GRAPE) is a popular technique in quantum optimal control, and can be combined with automatic differentiation (AD) to facilitate on-the-fly evaluation of cost-function gradients. We illustrate that the convenience of AD comes at a significant memory cost due to the cumulative storage of a large number of states and propagators. For quantum systems of increasing Hilbert space size, this imposes a significant bottleneck. We revisit the strategy of hard-coding gradients in a scheme that fully avoids propagator storage and significantly reduces memory requirements. Separately, we present improvements to numerical state propagation to enhance runtime performance. We benchmark runtime and memory usage and compare this approach to AD-based implementations, with a focus on pushing towards larger Hilbert space sizes. The results confirm that the AD-free approach facilitates the application of optimal control for large quantum systems which would otherwise be difficult to tackle.

I. INTRODUCTION

The rapidly evolving field of quantum information processing has generated much interest in quantum optimal control for tasks such as state preparation [1], error correction [2, 3] and realization of logical gates [4–7]. This methodology has been implemented in various quantum systems [8], such as nuclear magnetic resonance [9–14], trapped ions [15, 16], nitrogen-vacancy centers in diamonds [17–23], Bose-Einstein condensation [24–27] and neutral atoms [28, 29].

One of the prominent techniques for implementing quantum optimal control is Gradient Ascent Pulse Engineering (GRAPE). The central goal of optimal control is to adjust control parameters in such a way that the infidelity of the desired state transfer or gate operation is minimized. In GRAPE, the minimization is based on gradient descent and thus generally requires the evaluation of gradients. Automatic differentiation (AD) [30] is a convenient tool that has also made an impact on quantum optimal control with GRAPE [31–33]. Internally, AD builds a computational graph and applies the chain rule to automatically compute the gradient of a given function.

However, the convenience of AD comes at the cost of memory required for storing the full computational graph. The use of semi-automatic differentiation (semi-AD) [34] partially reduces the memory overhead compared to full automatic differentiation (full-AD). This reduction can still be insufficient to tackle optimal control for quantum systems of rapidly growing size [35, 36]. This motivates us to revisit the original GRAPE scheme [11] which is based on hard-coded gradients (HG), and has the smallest possible memory cost. We perform a scaling analysis of memory usage and runtime, and thereby develop a decision tree guiding the optimal choice of strategy among HG, semi-AD and full-AD. We exemplify the scaling behavior with benchmarks for concrete optimization tasks.

The outline of our paper is as follows. Section II briefly reviews GRAPE and introduces necessary notation. In Sec. III, we illustrate derivation and memory-efficient implementation of the gradients for the most commonly used cost func-

tion contributions. Our numerical implementation of HG further improves the efficiency of state propagation. In Sec. IV, we analyze and compare the scaling of runtime and memory usage scaling for HG, AD and semi-AD, and present results from concrete benchmark studies. In Sec. V, we formulate a decision tree that facilitates the choice of optimal numerical strategy for GRAPE-based quantum optimal control. We summarize our findings and present our conclusions in Sec. VI.

II. BRIEF SKETCH OF GRAPE

We briefly sketch the basics of GRAPE [11], mainly to establish the notion used in subsequent sections.

Consider a system described by the Hamiltonian

$$H(t) = H_s + a(t) h_c. \quad (1)$$

Here, H_s denotes the static system Hamiltonian and h_c is an operator coupling the classical control field $a(t)$ to the system¹. Quantum optimal control aims to adjust $a(t)$ such that a desired unitary gate or state transfer is performed within a given time interval $0 \leq t \leq T$. To facilitate optimization, the total control time T is divided into N intervals of duration $\Delta t = T/N$, and the control field is specified by the amplitudes at the discrete times $t_n = n\Delta t$ ($n = 1, 2, \dots, N$). The discretization interval Δt is chosen such that control amplitudes are approximately constant² within each Δt . We denote the time-discretized values by

$$a_n := a(t_n), \quad (2)$$

and group these to form the vector $\mathbf{a} \in \mathbb{R}^N$. Under the influence of this piecewise-constant control field, the system's

¹ For simplicity, we consider one control channel with real-valued $a(t)$. Generalizations to multiple control channels and complex a are straightforward.

² This treatment is actually quite close to modeling the actual output of an arbitrary waveform generator, where Δt can be chosen to align with the available resolution.

量子状態 $|\phi_n i := |\psi(t_n) i$ は、以下のように1つの時間ステップで進化する。

$$|\psi_n\rangle = U_n |\psi_{n-1}\rangle. \quad (3)$$

ここで、 U_n は短時間伝搬体である

$$U_n := U(t_n, t_{n-1}) = e^{-iH_n\Delta t} \quad (\hbar = 1), \quad (4)$$

ここで、 $H_n = -i(H_s + a_n h_c) \text{at}$ 。通常、Cには状態遷移やゲート不実性、さらに最大電力や帯域幅などのパルス特性を調整するための追加コストが含まれます。ここで、 α_v は経験的に選ばれた重み係数、 C_v は個々のコスト貢献度です。GRAPEは、コスト関数の勾配に従って制御振幅を更新することにより、C(a)を最小化する最急降下法を利用する：

$$\mathbf{a}^{(j+1)} = \mathbf{a}^{(j)} - \eta_j \nabla_{\mathbf{a}} C(\mathbf{a}^{(j)}). \quad (5)$$

ここで、 j は最急降下反復を列挙し、 $\eta_j > 0$ はステップサイズを支配する学習率である。

III. CALCULATION OF GRADIENTS

GRAPEにとって重要な要素は、コスト関数の勾配を計算することである。この目的のために、Tensorflow [37] や Autograd [38] に実装されているような自動微分を活用することが一般的な選択肢の一つです。この戦略の明確な利点は、複雑なコスト関数の勾配を実行時に自動的に評価することが容易であることです。しかし、この利便性は、大規模な量子システムで禁止される可能性のあるメモリ消費の代償として購入されます。このようなメモリ資源の浪費は、大規模な量子系に最適制御を適用する場合、非常に大きな問題となります。そこで、計算効率の良い方式でハードコードされた勾配を見直すことで、主要なコストに貢献することを動機としています。

A. Analytical gradients

この小節では、2つの主要なコスト貢献の解析的な勾配について説明する。他のコストの勾配を表す式はAp p. Aで示す。状態遷移問題において、量子最適制御の中心的な目標は、以下の式で与えられる状態遷移の不忠実性を最小化することである。

$$C_1^{st} = 1 - |\langle\phi_T|\psi_N\rangle|^2, \quad (6)$$

ここで、 $|\phi_T i$ は目標状態、 $|\phi_N i$ は最終時刻 t_N に実現した状態である。 C_1^{st} の勾配は次のような形になる。

$$\begin{aligned} \frac{\partial C_1^{st}}{\partial a_n} &= -\langle\phi_T|U_N \cdots \frac{\partial U_n}{\partial a_n} \cdots U_1|\psi_0\rangle\langle\psi_N|\phi_T\rangle - \text{c. c.} \\ &= -\langle\phi_n|\frac{\partial U_n}{\partial a_n}|\psi_{n-1}\rangle\langle\psi_N|\phi_T\rangle - \text{c. c.}, \end{aligned} \quad (7)$$

ここで、状態 $|\phi_n i$ は再帰的な関係に従う。

$$|\phi_n\rangle = U_{n+1}^\dagger |\phi_{n+1}\rangle. \quad (8)$$

これは逆伝播と解釈できる。状態遷移に加え、ユニタリーゲートも注目される操作の一つです。ゲートの最適化は、ゲート不貞度 $C_1^g = 1 - \text{tr } U_T^\dagger U_R / d^2$ を最小化することで実現できる。ここで、 U_T は目標ユニタリーゲート、 $U_R = U_N \cdots U_1$ は実際に実現するゲートを指す。与えられた正規直交基底 $\{\psi | \frac{h}{0} i\}_h$ (h は基底状態を列挙) に対して、 C_1^g の勾配は以下のように書くことができる。

$$\frac{\partial C_1^g}{\partial a_n} = -\frac{1}{d^2} \sum_h \langle\phi_n^h|\frac{\partial U_n}{\partial a_n}|\psi_{n-1}^h\rangle \sum_{h'} \langle\psi_N^{h'}|\phi_N^{h'}\rangle - \text{c. c.}, \quad (9)$$

ここで、 $|\phi_n^h i = U_n \cdots U_1 |\psi_0^h i$ とする。状態 $|\phi hn i$ は、基底状態 $\{\psi | \frac{h}{0} i\}_h$ に目標ユニタリー U_T を適用し、その結果を時刻 t_n に逆伝播することにより得られる、

$$|\phi_n^h\rangle = U_{n+1}^\dagger \cdots U_N^\dagger U_T |\psi_0^h\rangle. \quad (10)$$

式(7)と式(9)は、勾配降下に必要な解析式を提供する。次の2つのサブセクションでは、これらの式をメモリ効率の良い方法で数値的に評価する方法について説明します。

B. ハードコードされた勾配の数値的な実装

式(7)、(9)の勾配の計算には、 $U_n | \Psi i, \frac{\partial U_n}{\partial a_n} | \Psi i$ の形の式の数値評価が必要です。ここで、短時間伝搬体 U_n は、行列指数 e^A と $A = -iH_n \text{ at}$ によって与えられる。 $e^A | \Psi i$ を数値的に評価することは、困難がないわけではない。Moler and Van Loan [39]が指摘するように、いくつかの方法が存在するが、いずれも真に最適な方法ではない。参考文献[39]で上位にランクされている3つの方法は、常微分方程式の解法、行列の対角化、スケーリングと二乗（系列展開との組み合わせ）に基づくものです。この中で、常微分方程式を解くことは、実行時間が最も高くなる傾向があります[34]。我々は主にスケーリングとスクエアリングに焦点を当てますが、行列の対角化が望ましい状況については、Sec. Vでコメントします。

$$e^A = (e^{A/s})^s, \quad (11)$$

ここで、右辺の行列指数には、 $| | A | |$ に比べてノルムが小さくなった行列 $B = A/s$ が含まれるようになりました。これにより、一般に指数級数³の切り捨て次数 m を低くすることができる、

$$e^B \approx e_m^B := \sum_{q=0}^m \frac{B^q}{q!}. \quad (12)$$

³ Chebyshev展開はTaylor展開より収束が早いですが、両者の数値実装は同じ実行時間スケーリングとなります。ここでは、その単純性からTaylor展開に焦点を当てます。

quantum state $|\psi_n\rangle := |\psi(t_n)\rangle$ evolves by a single time step according to

$$|\psi_n\rangle = U_n |\psi_{n-1}\rangle. \quad (3)$$

Here, U_n is the short-time propagator

$$U_n := U(t_n, t_{n-1}) = e^{-iH_n\Delta t} \quad (\hbar = 1), \quad (4)$$

where $H_n = -i(H_s + a_n h_c)\Delta t$. Optimization of the control field proceeds via minimization of a cost function C . Typically, C includes the state-transfer or gate infidelity, along with additional cost contributions employed to moderate pulse characteristics such as maximal power or bandwidth. To this end a composite cost function is constructed, $C(\mathbf{a}) = \sum_\nu \alpha_\nu C_\nu(\mathbf{a})$, where α_ν are empirically chosen weight factors and C_ν are individual cost contributions. GRAPE utilizes the method of steepest descent to minimize $C(\mathbf{a})$ by updating the control amplitudes according to the cost function gradient:

$$\mathbf{a}^{(j+1)} = \mathbf{a}^{(j)} - \eta_j \nabla_{\mathbf{a}} C(\mathbf{a}^{(j)}). \quad (5)$$

Here, j enumerates steepest-descent iterations and $\eta_j > 0$ is the learning rate governing the step size.

III. CALCULATION OF GRADIENTS

A critical ingredient for GRAPE is the computation of cost function gradients. One popular option is to leverage automatic differentiation for this purpose, such as implemented in Tensorflow [37] and Autograd [38]. A clear advantage of this strategy is the ease by which complicated cost function gradients are evaluated automatically at runtime. This convenience, however, is bought at the cost of significant memory consumption which can be prohibitive for large quantum systems. This waste of memory resources is prohibitive when optimal control is applied to large quantum systems. Therefore, we are motivated to revisit hard-coded gradients in a computationally efficient scheme for key cost contributions.

A. Analytical gradients

We provide a discussion of the analytical gradients of two key cost contributions in this subsection. Expressions for gradients of other cost contributions are shown in App. A.

In state-transfer problems, a central goal of quantum optimal control is to minimize the state-transfer infidelity given by

$$C_1^{st} = 1 - |\langle \phi_T | \psi_N \rangle|^2, \quad (6)$$

where $|\phi_T\rangle$ is the target state and $|\psi_N\rangle$ is a state realized at final time t_N . The gradient of C_1^{st} takes the form

$$\begin{aligned} \frac{\partial C_1^{st}}{\partial a_n} &= -\langle \phi_T | U_N \cdots \frac{\partial U_n}{\partial a_n} \cdots U_1 | \psi_0 \rangle \langle \psi_N | \phi_T \rangle - \text{c. c.} \\ &= -\langle \phi_n | \frac{\partial U_n}{\partial a_n} | \psi_{n-1} \rangle \langle \psi_N | \phi_T \rangle - \text{c. c.}, \end{aligned} \quad (7)$$

where the state $|\phi_n\rangle$ obeys the recursive relation

$$|\phi_n\rangle = U_{n+1}^\dagger |\phi_{n+1}\rangle. \quad (8)$$

This can be interpreted as backward propagation.

In addition to state transfer, unitary gates are another operation of interest. Gate optimization can be achieved by minimizing the gate infidelity $C_1^g = 1 - |\text{tr}(U_T^\dagger U_R)/d|^2$. Here, U_T is the target unitary gate and $U_R = U_N \cdots U_1$ is the actually realized gate. For a given orthonormal basis $\{|\psi_0^h\rangle\}_h$ (h enumerates the basis states), the gradient of C_1^g can be written as

$$\frac{\partial C_1^g}{\partial a_n} = -\frac{1}{d^2} \sum_h \langle \phi_n^h | \frac{\partial U_n}{\partial a_n} | \psi_{n-1}^h \rangle \sum_{h'} \langle \psi_N^{h'} | \phi_N^{h'} \rangle - \text{c. c.}, \quad (9)$$

where $|\psi_n^h\rangle = U_n \cdots U_1 |\psi_0^h\rangle$. The state $|\phi_n^h\rangle$ is obtained as follows: apply the target unitary U_T to basis state $\{|\psi_0^h\rangle\}_h$ and backward propagate the result to time t_n ,

$$|\phi_n^h\rangle = U_{n+1}^\dagger \cdots U_N^\dagger U_T |\psi_0^h\rangle. \quad (10)$$

Equations (7) and (9) provide the analytical expressions needed for gradient descent. In the next two subsections, we will discuss how to numerically evaluate these expressions in a memory-efficient way.

B. Numerical implementation of hard-coded gradient

The calculation of gradients in Eqs. (7) and (9) requires numerical evaluation of expressions of the form $U_n |\Psi\rangle$ and $\frac{\partial U_n}{\partial a_n} |\Psi\rangle$. Here, the short-time propagator U_n is given by a matrix exponential e^A and $A = -iH_n\Delta t$.

Numerically evaluating $e^A |\Psi\rangle$ is not without challenge. As pointed out by Moler and Van Loan [39], several methods exist, yet none of them is truly optimal. Three methods ranked highly in reference [39] are based on solving ordinary differential equations, matrix diagonalization and scaling and squaring (combined with series expansion). ODE solving, in this context, tends to be most expensive in runtime [34]. We mainly focus on scaling and squaring but comment on situations where matrix diagonalization is preferable in Sec. V.

Scaling and squaring [40] which is based on the identity

$$e^A = (e^{A/s})^s, \quad (11)$$

where the right hand side matrix exponential now involves the matrix $B = A/s$ which has a norm that is reduced compared to $\|A\|$. This generally allows for a lower truncation order m of the exponential series³,

$$e^B \approx e_m^B := \sum_{q=0}^m \frac{B^q}{q!}. \quad (12)$$

³ While Chebyshev expansion has faster convergence than Taylor expansion, their numerical implementations have the same runtime scaling. Here, we focus on Taylor expansion due to its simplicity.

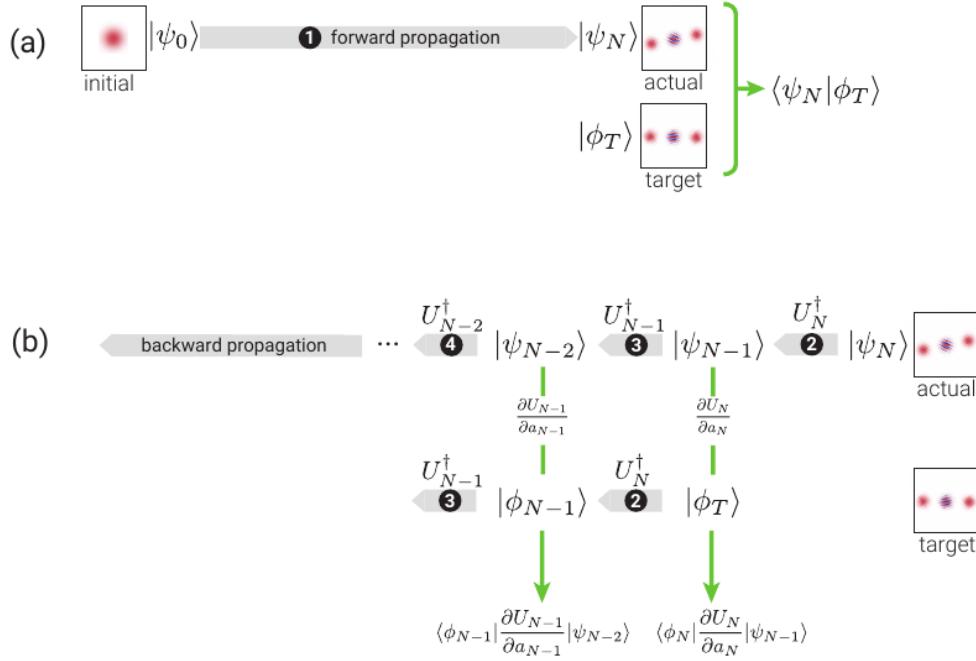


図1. 式(7)の勾配の成分を計算するための前方-後方伝搬スキームである。(a) 初期状態を $U_{N-2} \dots U_1$ で順伝播して実現状態 $|\psi_N\rangle$ を得、オーバーラップ $\langle\psi_N|\phi_T\rangle$ (ϕ_T は目標状態) を算出する。(b) 伝搬子 U_N^\dagger 、 U_{N-1}^\dagger によって $|\psi_N\rangle$ 、 $|\phi_T\rangle$ を後方伝播させる。 \dots 、 U_1^\dagger を繰り返し、各ステップ段階で $\frac{\partial U_n}{\partial a_n}$ を算出する。

通常、 m は元の e^A に必要なものよりも小さいです。我々の目的は、状態ベクトル $e^A |\Psi\rangle \approx (e_m^B)^s |\Psi\rangle$ を近似することである。

$$\begin{aligned} e_m^B |\Psi\rangle &= \left(1 + B + \frac{B^2}{2!} + \dots + \frac{B^m}{m!}\right) |\Psi\rangle \\ &= |\Psi\rangle + B|\Psi\rangle + \frac{B(B|\Psi\rangle)}{2} + \dots + \frac{B(\dots(B|\Psi\rangle))}{m!}. \end{aligned} \quad (13)$$

最後の式により、 $e^A |\Psi\rangle$ の近似は、行列と行列の乗算を一度も行わず、ベクトルへの $B = A/s$ の繰り返し適用により進められる。これにより、実行性能は $O(d^3)$ から $O(d^2)$ に向上する。 $e^A |\Psi\rangle$ の評価を進めるために、誤差が誤差許容範囲 τ 以下になるように切り捨て順序 m とスケーリングファクター s を決定する、

$$\varepsilon_{A,|\Psi\rangle}(m, s) := \frac{\|e^A |\Psi\rangle - [(e_m^B)^s |\Psi\rangle]\|}{\|e^A |\Psi\rangle\|} < \tau. \quad (14)$$

ここで、 $-$ の浮動小数点表現を示す。この誤差 $\varepsilon_{A,|\Psi\rangle}$ は評価が難しくコストがかかるので、代わりに上界 $E_A(m, s)$ [A pp. C 参照]を導き、不等式に頼ります。

$$\varepsilon_{A,|\Psi\rangle}(m, s) < E_A(m, s) \leq \tau. \quad (15)$$

与えられた A に対して、この不等式は一般に m と s に対して複数の解を持つ。Cに示すように、この評価には m の行列-ベクトル乗算が必要であり、これが実行時間を支配していることが観察されている。

m s の系統的な最小化にはかなりの実行時間が必要であるため、代わりに以下のことを行う：(1) 式(15)の解のうち、 s が最小のものを選び (s と表記)、(2) これらの組のうち、 m が最小のものを選ぶ (m と表記する)。したがって、 $e^A |\Psi\rangle$ を計算するためには

$$\mu := m s \quad (16)$$

行列-ベクトル乗算を行う。App. Dで2つの具体例で示すように Dで2つの具体例を示すように、 $e^A |\Psi\rangle$ を評価する我々の方法は、Scipyのexpm_multiply関数[41]よりも優れた実行性能と精度を達成できる。 $e^A |\Psi\rangle$ の数値評価に関する課題を解決した後、式(7)と(9)のようにコスト関数勾配を計算するために重要な $\frac{\partial U_n}{\partial a_n} |\Psi\rangle$ の計算について説明します。この評価は、近似式

$$\frac{\partial U_n}{\partial a_n} |\Psi\rangle \approx \frac{\partial (e_m^{B_n})^s}{\partial a_n} |\Psi\rangle, \quad (17)$$

ここで、 $B_n = -iH_n \Delta t / s$ とする。恒等式[42, 43]に基づく補助行列法を用いて進める。

$$(e_m^{\mathcal{A}_n/s})^s \begin{pmatrix} 0 \\ |\Psi\rangle \end{pmatrix} = \begin{pmatrix} \frac{\partial (e_m^{B_n})^s}{\partial a_n} |\Psi\rangle \\ (e_m^{B_n})^s |\Psi\rangle \end{pmatrix}. \quad (18)$$

これは、 $(\partial (e_m^B)^s / \partial a_n) |\Psi\rangle$ とベクトルに作用する補助行列 A_n の指標との関係である。ここで、 A_n は次のように定義される。

$$\mathcal{A}_n := \begin{pmatrix} -iH_n \Delta t & -iH_c \Delta t \\ 0 & -iH_n \Delta t \end{pmatrix}, \quad (19)$$

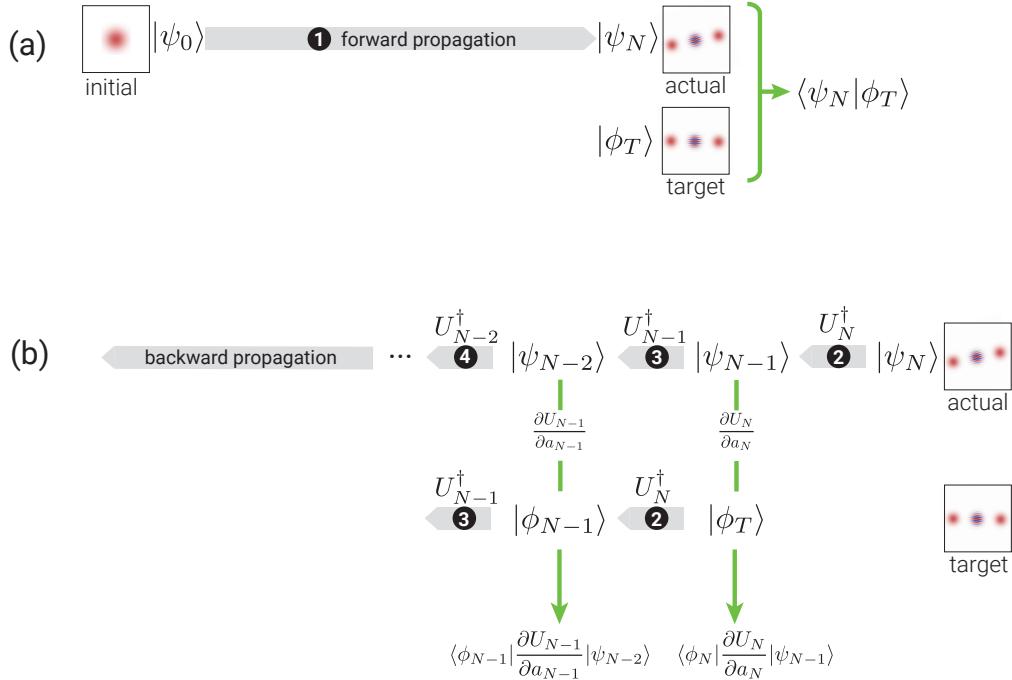


Figure 1. The forward-backward propagation scheme to compute the components of the gradient in Eq. (7). (a) Forward propagate the initial state by $U_N \cdots U_1$ to obtain the realized state $|\psi_N\rangle$ and calculate the overlap $\langle\psi_N|\phi_T\rangle$, where $|\phi_T\rangle$ is the target state. (b) Backward propagate $|\psi_N\rangle$ and $|\phi_T\rangle$ by the propagators $U_N^\dagger, U_{N-1}^\dagger, \dots, U_1^\dagger$ iteratively, and compute $\langle\phi_n|\frac{\partial U_n}{\partial a_n}|\psi_{n-1}\rangle$ at each step step.

Typically, m is smaller than the one required for the original e^A . Our goal is to approximate the state vector $e^A|\Psi\rangle \approx (e_m^B)^s|\Psi\rangle$ where

$$\begin{aligned} e_m^B|\Psi\rangle &= \left(1 + B + \frac{B^2}{2!} + \cdots + \frac{B^m}{m!}\right)|\Psi\rangle \quad (13) \\ &= |\Psi\rangle + B|\Psi\rangle + \frac{B(B|\Psi\rangle)}{2} + \cdots + \frac{B(\cdots(B|\Psi\rangle))}{m!}. \end{aligned}$$

According to the last expression, approximation of $e^A|\Psi\rangle$ can thus proceed by repeated application of $B = A/s$ to a vector, without ever invoking matrix-matrix multiplication. This boosts runtime performance from $O(d^3)$ to $O(d^2)$.

To proceed with the evaluation of $e^A|\Psi\rangle$, the truncation order m and scaling factor s are determined in such a way that the error remains below the error tolerance τ ,

$$\varepsilon_{A,|\Psi\rangle}(m, s) := \frac{\|e^A|\Psi\rangle - \llbracket(e_m^B)^s|\Psi\rangle\rrbracket\|}{\|e^A|\Psi\rangle\|} < \tau. \quad (14)$$

Here, $\llbracket \bullet \rrbracket$ denotes the floating point representation of \bullet . Since this error $\varepsilon_{A,|\Psi\rangle}$ is difficult and costly to evaluate, we instead derive an upper bound $E_A(m, s)$ [see App. C] and resort to the inequality

$$\varepsilon_{A,|\Psi\rangle}(m, s) < E_A(m, s) \leq \tau. \quad (15)$$

For a given A , this inequality generally has multiple solutions for m and s . As shown in App. C, the evaluation requires ms matrix-vector multiplications, which are observed to dominate

the runtime. Since systematic minimization of ms requires significant runtime itself, we instead: (1) select solutions to Eq. (15) with the smallest s which is denoted by \mathbb{s} , and (2) among these pairs, choose the one with smallest m (which we denote by \mathbb{m}). Thus, calculating $e^A|\Psi\rangle$ requires

$$\mu := \mathbb{m}\mathbb{s} \quad (16)$$

matrix-vector multiplications. As shown in App. D for two concrete examples, our method for evaluating $e^A|\Psi\rangle$ can achieve better runtime performance and accuracy than Scipy's `expm_multiply` function [41].

Having addressed the challenges associated with numerically evaluating $e^A|\Psi\rangle$, we now turn to the calculation of $\frac{\partial U_n}{\partial a_n}|\Psi\rangle$, which is crucial for computing cost function gradients as in Eqs. (7) and (9). We base this evaluation on the approximation

$$\frac{\partial U_n}{\partial a_n}|\Psi\rangle \approx \frac{\partial(e_m^{B_n})^s}{\partial a_n}|\Psi\rangle, \quad (17)$$

where $B_n = -iH_n\Delta t/s$. We proceed by using auxiliary matrix method based on the identity [42, 43]

$$(e_m^{\mathcal{A}_n/s})^s \begin{pmatrix} 0 \\ |\Psi\rangle \end{pmatrix} = \begin{pmatrix} \frac{\partial(e_m^{B_n})^s}{\partial a_n}|\Psi\rangle \\ (e_m^{B_n})^s|\Psi\rangle \end{pmatrix}. \quad (18)$$

This relates $(\partial(e_m^{B_n})^s/\partial a_n)|\Psi\rangle$ to the exponential of the auxiliary matrix \mathcal{A}_n acting on a vector. Here, \mathcal{A}_n is defined as

$$\mathcal{A}_n := \begin{pmatrix} -iH_n\Delta t & -ih_c\Delta t \\ 0 & -iH_n\Delta t \end{pmatrix}, \quad (19)$$

表1. ヒルベルト空間次元d, 時間ステップ数N, 格納されたハミルトニアン行列要素 $\kappa = \kappa(d)$, 伝搬体-状態積の評価に必要な行列ベクトル乗算数 $\mu = \mu(d)$ に対するメモリ使用量とランタイムスケーリング。(μ と κ のスケーリングは特定の最適化タスクに依存します。) 表は、ハードコード勾配(HG)、全自动微分(full-AD)、半自动微分(semi-AD)に対する状態転送とゲート動作のスケーリングを対比しています。

	STATE TRANSFER		GATE OPERATION	
	Runtime	Memory usage	Runtime	Memory usage
Hard-coded gradient	$\Theta(N\mu\kappa)$	$\Theta(d + \kappa)$	$\Theta(N\mu\kappa d)$	$\Theta(d + \kappa)$
Full automatic differentiation	$\Theta(N\mu\kappa)$	$\Theta(N\mu d + \kappa)$	$\Theta(N\mu\kappa d)$	$\Theta(N\mu d^2)$
Semi-automatic differentiation	$\Theta(N\mu\kappa)$	$\Theta(Nd + \kappa)$	$\Theta(N\mu\kappa d)$	$\Theta(Nd^2)$

A_n は、次元dのヒルベルト空間に作用する演算子の 2×2 行列である。

C. ハードコードされた勾配を効率的に評価する。

式(7)の ∇C_1^{st} のような解析的な勾配式は、一般に複雑で、多くの寄与量を含んでいます。演算のグループ化や順序によつては、これらの量を同時に保存したり、繰り返し計算したりする必要があり、実行時間やメモリ使用量に重大な影響を与えることがある。Khanejaら[11]が提案した方式は、保存された中間状態ベクトルの拡散を避けるという、好ましい妥協点を突いている。具体的には、最終的な量子状態 $|\psi_N\rangle_i$ は、 $|\psi_0\rangle_i$ を順伝播することで得られる。次に、 $|\psi_N\rangle_i$, $|\phi_T\rangle_i$ の後方伝播を行い、勾配のn番目の成分に必要な行列要素 $h\phi_n - |\phi\partial U_n\rangle_i$ を計算し、勾配を降順に積み上げる。両伝搬方向とも、現在の中間状態のみが記憶されるため、メモリコストは $\Theta(1)$.⁴となる。

IV. SCALING ANALYSIS AND BENCHMARKING

GRAPEで勾配を計算する方法として、ハードコードされた勾配の評価、全自动微分[31]、半自动微分[34]の3つを検討しました。与えられた問題に最も適した方法を決定するために、これら3つのアプローチのメモリ使用量と実行時間のスケーリングを調査する。この分析に続いて、特定の状態遷移とゲートオペレーション問題のベンチマークスタディを提示し、スケーリング挙動を説明する。

A. ランタイムとメモリ使用量のスケーリング分析

ヒルベルト空間次元(d)が大きい量子系や、時間ステップ数(N)が大きい時間発展系の最適化を行う場合、ランタイムやメモリ使用量が大きな問題となることがあります。本節では、状態遷移とユニタリゲート最適化のコスト寄与度 C_1 に着目し、Nとdに対するメモリ使用量とランタイムのスケーリングを分析する。この結果は、 C_1 以外のコスト貢献にも一般化することができます。App.Aで詳しく説明する。a. 前後伝搬を用いたハードコード勾配(HG)の評価。まず、状態遷移の問題を考える。 C_1^{st} を評価するためのメモリ使用量は、主に量子状態とハミルトニアンの必要なストレージによって決まります。前方後方伝搬では、任意の時間に格納される状態とハミルトニアンマトリクスのインスタンスの数は、1つのオーダーである。d次元の個々の状態ベクトルは、メモリ $\Theta(d)$ を消費する。ハミルトニアンのメモリ使用量は、採用された記憶方式に依存する。密行列ストレージの「最悪」ケースでは、ハミルトニアンに必要なメモリは $\Theta(d^2)$ です。スペース行列ストレージの場合、メモリ使用量は一般にスペース性に依存し、格納されたハミルトニアン行列要素数（ここでは κ と表記）によって決定される。例えば、対角行列や三角行列の場合、メモリ使用量は $\kappa \Theta(d)$ 、最近接 σ_z 結合を持つ量子ビットの線形鎖の場合は $\Theta(d \log_2 d)$ です。上記のスケーリング関係を組み合わせることで、メモリ使用量の全体的な漸近挙動 $\Theta(d + \kappa)$ が得られます。このスキームの実行時間は、伝搬体状態の積とその微分の対応するものの必要な $\Theta(N)$ 個の評価によって支配されていることがわかります。1つの伝搬体状態積を数値的に近似するには、 μ 個の行列ベクトル乗算[式(16)]が必要であり、行列ベクトル乗算の1インスタンスのランタイムは $\Theta(\kappa)$ としてスケールします。 $\mu = \mu(d)$ のdに対する暗黙の依存性は、それぞれの特定のシステムに特有のものであり、解析的に求めることは困難である場合があります。数値的には、空洞に結合した駆動トランスマジコンに対して $\mu \Theta(d)$ 、結合した量子ビットの線形チェーンに対して $\Theta(10 g_2 d)$ のスケーリングが得られることがわかった。詳細はApp.Eを参照。上記の個々のスケーリング関係を組み合わせると、全体的な実行時間の漸近挙動 $\Theta(N \mu \kappa)$ につながります。

⁴ ここで、 $f(x) = \Theta(g(x))$ は、漸近的な意味で $g(x)$ によって上下に拘束される $f(x)$ の通常のバッハマン・ランダウ表記である。

Table I. Memory usage and runtime scaling with respect to Hilbert space dimension d , number of time steps N , stored Hamiltonian matrix elements $\kappa = \kappa(d)$ and the number of matrix-vector multiplications required for evaluating propagator-state products $\mu = \mu(d)$. (The scaling of μ and κ depends on the specific optimization task.) The table contrasts scaling for state transfer and gate operation for hard-coded gradients (HG), full automatic differentiation (full-AD) and semi-automatic differentiation (semi-AD).

	STATE TRANSFER		GATE OPERATION	
	Runtime	Memory usage	Runtime	Memory usage
Hard-coded gradient	$\Theta(N\mu\kappa)$	$\Theta(d + \kappa)$	$\Theta(N\mu\kappa d)$	$\Theta(d + \kappa)$
Full automatic differentiation	$\Theta(N\mu\kappa)$	$\Theta(N\mu d + \kappa)$	$\Theta(N\mu\kappa d)$	$\Theta(N\mu d^2)$
Semi-automatic differentiation	$\Theta(N\mu\kappa)$	$\Theta(Nd + \kappa)$	$\Theta(N\mu\kappa d)$	$\Theta(Nd^2)$

which is a 2×2 matrix of operators each acting on our Hilbert space with dimension d . Once a basis is chosen, \mathcal{A}_n can be represented as a $2d \times 2d$ matrix of complex numbers.

C. Efficient evaluation of hard-coded gradients

The analytical gradient expressions, such as ∇C_1^{st} in Eq. (7), are generally complicated and involve a large number of contributing quantities. Depending on grouping and ordering of operations, these quantities may need to be stored simultaneously or computed repeatedly with critical implications for runtime and memory usage. The scheme originally proposed by Khaneja *et al.* [11] strikes a favorable compromise that avoids the proliferation of stored intermediate state vectors. More specifically, the final quantum state $|\psi_N\rangle$ is obtained by forward propagating $|\psi_0\rangle$. Then, backward propagation of $|\psi_N\rangle, |\phi_T\rangle$ is carried out to calculate the matrix element $\langle \phi_n | \frac{\partial U_n}{\partial a_n} | \psi_{n-1} \rangle$ needed for the n -th component of the gradient, thus building up the gradient in descending order. For both propagation directions, only the current intermediate states are stored, leading to a memory cost of $\Theta(1)$.⁴

IV. SCALING ANALYSIS AND BENCHMARKING

We consider the following three methods to compute gradients within GRAPE: evaluation of hard-coded gradients, full automatic differentiation [31], semi-automatic differentiation [34]. To determine the method most appropriate for a given problem, we inspect the memory usage and runtime scaling of these three approaches. Following this analysis, we present benchmark studies of specific state-transfer and gate-operation problems to illustrate the scaling behavior.

A. Scaling analysis of runtime and memory usage

Runtime and memory usage can pose significant challenges when performing optimization tasks for quantum systems with large Hilbert space dimension (d) or for time evolution requiring significant number of time steps (N). In this subsection, we analyze the scaling of memory usage and runtime with respect to both N and d , focusing on the cost contribution C_1 for state-transfer and unitary-gate optimization. Our findings can be generalized to other cost contributions beyond C_1 , as elaborated in App. A.

a. *Evaluation of hard-coded gradients (HG) using forward-backward propagation.* We start by considering state-transfer problems. The *memory usage* for evaluating ∇C_1^{st} is mainly determined by the required storage of quantum states and the Hamiltonian. In forward-backward propagation, the number of states and instances of Hamiltonian matrices stored at any given time is of the order of unity. Each individual state vector of dimension d consumes memory $\sim \Theta(d)$. Memory usage for the Hamiltonian depends on the employed storage scheme. In the “worst” case of dense matrix storage, the memory required for Hamiltonian is $\Theta(d^2)$. For sparse matrix storage, memory usage generally depends on sparsity and is determined by the number of stored Hamiltonian matrix elements (here denoted as κ). For example, for a diagonal or tridiagonal matrix, memory usage is $\kappa \sim \Theta(d)$; for a linear chain of qubits with nearest-neighbor σ_z coupling, it is $\Theta(d \log_2 d)$. Combining the scaling relations above gives us an overall asymptotic behavior of memory usage $\Theta(d + \kappa)$.

We observe that *runtime* of the scheme is dominated by the required $\Theta(N)$ evaluations of propagator-state products and their derivative counterparts. Numerically approximating a single propagator-state product requires μ matrix-vector multiplications [see Eq. (16)], where runtime of one instance of matrix-vector multiplication scales as $\Theta(\kappa)$. The implicit dependence of $\mu = \mu(d)$ on d is specific to each particular system and can be difficult to obtain analytically. Numerically, we find for a driven transmon coupled to a cavity that $\mu \sim \Theta(\sqrt{d})$; for a linear chain of coupled qubits, the scaling is $\Theta(\log_2 d)$, see App. E for more details. Once combined, the above individual scaling relations lead to an overall

⁴ Here, $f(x) = \Theta(g(x))$ is the usual Bachmann–Landau notation for $f(x)$ bounded above and below by $g(x)$ in the asymptotic sense.

例えば、結合量子ビットの線形鎖の場合、実行時間は $\Theta(N d (\log_2 d)^2)$ のようにスケールします。次に、ゲート最適化問題に目を向けると、メモリ使用量とランタイムのスケーリング挙動は次の通りである。d個の基底状態のそれぞれに対して順方向-逆方向伝搬を順次適用して、 ∇C_1^g を計算する。メモリ使用量は $\Theta(d + \kappa)$ で、 ∇C_1^{st} の場合と同じである。実行時間のスケーリングは $\Theta(N \mu \kappa d)$ で、 ∇C_1^{st} の実行時間のスケーリングのd倍である。⁵ b. 全自動微分(full-AD) 逆モード自動微分は、計算木を順パスと逆パスで勾配を取り出す。フォワードパスではコスト関数Cが計算され、バックワードパスではチェーンルールにより勾配が累積される。 C_1^{st} を評価するためのメモリ使用量は、すべての中間数値結果をフォワードパスの一部として保存し、リバースパスの入力として機能しなければならないという要件によって決定される。具体的には、フォワードパスでは、N個の時間ステップで初期状態ベクトルを最終状態ベクトルに進化させる。スケーリングと二乗による短時間の伝搬状態積の計算には μ 回の反復が必要であるため、この過程で保存しなければならないベクトルの総数は $\Theta(N \mu)$ である。また、ハミルトニアンのメモリ使用量を考慮すると、完全なスケーリングは $\Theta(N \mu d + \kappa)$ となります。 C_1^g を評価するためのランタイムは、フォワードパスとリバースパスを組み合わせたものである。これは、 $\Theta(N)$ 個の伝搬体状態積の評価によって支配され、その各々はランタイムスケーリング $\Theta(\mu \kappa)$ を持ち、全体のランタイムスケーリングは $\Theta(N \mu \kappa)$ となる。ゲート最適化問題では、 C_1^g は単一の初期状態だけでなく、d個の基底ベクトルのセット全体を進化させることで評価される。そのため、メモリ使用量は状態遷移に比べてd倍増加し、全体としてメモリスケーリング $\Theta(N \mu d^2)$ となる。個々の基底ベクトルの進化が並列化されないと仮定すると、実行時間もd倍増加するため、 ∇C_1^g の実行時間スケーリングは $\Theta(N \mu \kappa d)$ となる。

c. 半自動微分(semi-AD) 分析的に導くのが面倒なコスト寄与の勾配に対しては、半自動微分[34]が、一部の微分をハードコードして、他の微分を自動処理するというアプローチを提案する。full-ADと比較して、semi-ADは、同じランタイムスケーリングを維持しながら、メモリ使用量のスケーリングが μ に依存しないという利点がある。d. 比較 このスケーリング分析の結果は、表Iに要約されている。フルADの場合、メモリ使用量は μ とステップ数Nに対して線形に成長する。これに対し、HGのメモリ使用量は μ とNに対して増加せず、一定に保たれる。この利点は、実行時スケーリングの悪化という代償を払うことなく得られる。したがって、HGは、メモリ使用量が法外に大きくなるような大規模量子系の最適制御において、full-ADよりも好ましいと言えます。

Semi-ADは、メモリボトルネックがそれほど深刻でない場合に実行可能な、興味深い妥協点を提示する。以下のセクションでは、具体的な最適化問題に対するベンチマークスタディによって、議論されたスケーリングについて説明する。

B. Benchmark studies

最初のベンチマーク例では、次のような状態遷移の問題を研究します。超伝導空洞に容量結合されたfluxtunable transmissionを駆動し、空洞の真空状態から20フォトンFock状態への状態遷移を行うという設定を考える。トランスマント駆動と共回転するフレームにおけるハミルトニアンは次の通りである。

$$H_1(t) = \Delta b^\dagger b + \frac{1}{2} \alpha b^\dagger b(b^\dagger b - 1) + g(cb^\dagger + c^\dagger b) + a_z(t)b^\dagger b + a_x(t)(b + b^\dagger). \quad (20)$$

ここで、駆動は縦方向と横方向の両方、cとbはそれぞれ空洞光子とトランスマント駆動のための消滅演算子である。 α 、 g 、 Δ は非調和性、相互作用の強さ、トランスマントと空洞の周波数の差を表す。最適化は、不純物 C_1^{st} を最小化し、コスト寄与 C_2^{st} を含むことでより高いトランスマントレベルの占有を制限することで進行する〔式(A1)〕。後者には、数個のトランスマントレベルへの切り捨てを可能にするという利点もある。我々は、1回の最適化反復のためのフルAD⁶とHGのランタイムとピークメモリ使用量を測定する。ランタイムはTemciパッケージ[44]で測定し、ピークメモリ使用量はメモリプロファイラパッケージ[45]で監視しています。最適化は、周波数3.7GHzのAMD Ryzen Threadripper 3970X 32-Core CPUで実行されています。状態遷移最適化のベンチマーク結果を図2に示します。単一の時間ステップ($N = 1$)を考慮し、dに対する実行時間を測定します。図2(a)の結果は、full-ADとHGがdに対して時間ステップあたりの実行時間のスケーリングが同じであることを確認します(表I)。観測された実行時間のスケーリング $\Theta(d^5/2)$ は、 H_1 の密なmatrixストレージによる $\kappa \Theta(d^2)$ と、 $\mu \Theta(d/2)$ (付録E参照)と一致する。メモリ使用量は、full-ADとHGの両方で、次元dに対して $\Theta(d^2)$ となり、 $\kappa \Theta(d^2)$ が支配的であることと矛盾しません。予想通り、Nに対するメモリ使用量のスケーリングはfull-ADとHGで異なる[図2(c)]。full-ADではNに対して線形であり、HGではNとは無関係である。このことから、時間ステップ数の多い最適化問題において、HGはfull-ADと比較してメモリ効率が良いことが確認された。

⁵ C_1^g を計算する別の方法は、すべての基底状態の並列化された前方後方伝搬からなります。この場合、具体的なスケーリング挙動は並列化スキームに依存するが、この議論は本論文の範囲外である。

⁶ ADは本ベンチマークではAutograd[38]というパッケージで実装されています。このパッケージはスパース線形代数をサポートしていないため、公平な比較のために、full-ADとHGの両方でハミルトニアンのための密集行列ストレージを実装しています。

runtime asymptotic behavior $\Theta(N\mu\kappa)$. For example, in the case of the linear chain of coupled qubits, runtime scales as $\Theta(Nd(\log_2 d)^2)$.

We next turn to gate-optimization problems, where scaling behavior of memory usage and runtime are as follows. We calculate ∇C_1^g by sequentially applying forward-backward propagation to each of the d basis states. The memory usage is $\Theta(d + \kappa)$, the same as for ∇C_1^{st} . The runtime scaling is $\Theta(N\mu\kappa d)$ which is d times larger than the runtime scaling for ∇C_1^{st} .⁵

b. Full automatic differentiation (full-AD) Reverse-mode automatic differentiation extracts gradients by a forward pass and a reverse pass through the computational tree. With the forward pass the cost function C is computed and the backward pass accumulates the gradient via chain rule.

The *memory usage* for evaluating ∇C_1^{st} is determined by the requirement that all intermediate numerical results must be stored as part of the forward pass and act as input to the reverse pass. Specifically, the forward pass evolves the initial state vector to the final state vector in N time steps. The total number of vectors that have to be stored along the way is $\Theta(N\mu)$, since computation of a single short-time propagator-state product via scaling and squaring necessitates μ iterations. In addition, considering the memory usage for the Hamiltonian, the full scaling is $\Theta(N\mu d + \kappa)$.

The *runtime* for evaluating ∇C_1^{st} combines forward and reverse pass. It is dominated by the evaluation of $\Theta(N)$ propagator-state products, each of which has the runtime scaling $\Theta(\mu\kappa)$, leading to an overall runtime scaling of $\Theta(N\mu\kappa)$.

For gate-optimization problems, C_1^g is evaluated by evolving not only a single initial state but a whole set of d basis vectors. Accordingly, memory usage increases by a factor of d relative to state transfer, resulting in the overall memory scaling $\Theta(N\mu d^2)$. Assuming that the evolution of individual basis vectors is not parallelized, runtime also grows by a factor of d , so that runtime scaling for ∇C_1^g is $\Theta(N\mu\kappa d)$.

c. Semi-automatic differentiation (semi-AD) For gradients of cost contributions that are tedious to derive analytically, semi-automatic differentiation [34] offers an alternative approach in which some derivatives are hard-coded and others treated by automatic differentiation. Compared to full-AD, semi-AD has the benefit that the memory usage scaling is independent of μ while maintaining the same runtime scaling.

d. Comparison The results of this scaling analysis are summarized in Tab. I. For full-AD, memory usage grows linearly with respect to μ and number of time steps N . By contrast, memory usage for HG does not increase with μ and N , but remains constant. This advantage is not bought at the cost of worse runtime scaling. Therefore, HG is preferable over full-AD in the optimal control of large quantum systems where memory usage would otherwise be prohibitively large.

Semi-AD presents an interesting compromise viable whenever the memory bottleneck is not too severe.

We illustrate the discussed scaling by benchmark studies for concrete optimization problems in the following section.

B. Benchmark studies

Our first benchmark example studies the following state-transfer problem. Consider a setup in which a driven flux-tunable transmon is capacitively coupled to a superconducting cavity, and perform a state transfer from the cavity vacuum state to the 20-photon Fock state. The Hamiltonian in the frame co-rotating with the transmon drive is

$$H_1(t) = \Delta b^\dagger b + \frac{1}{2}\alpha b^\dagger b(b^\dagger b - 1) + g(cb^\dagger + c^\dagger b) + a_z(t)b^\dagger b + a_x(t)(b + b^\dagger). \quad (20)$$

Here, the drive is both longitudinal and transverse, c and b are the annihilation operators for cavity photons and transmon excitations respectively. α , g and Δ denote anharmonicity, interaction strength and difference between the transmon and cavity frequency.

Optimization proceeds by minimizing the infidelity C_1^{st} and limiting the occupation of higher transmon levels by including the cost contribution C_2^{st} [Eq. (A1)]. The latter has the additional benefit of enabling the truncation to only a few transmon levels. We measure runtime and peak memory usage of full-AD⁶ and HG for a single optimization iteration. The runtime is measured by the package Temci [44] and the peak memory usage is monitored by the memory_profiler package [45]. The optimization is performed on an AMD Ryzen Threadripper 3970X 32-Core CPU with 3.7 GHz frequency.

Benchmark results for the state-transfer optimization are shown in Fig. 2. We consider a single time step ($N = 1$) and measure the runtime versus d . The results in Fig. 2(a) confirm that full-AD and HG have the same scaling of runtime per time step with respect to d (Tab. I). The observed runtime scaling $\Theta(d^{\frac{5}{2}})$ is consistent with $\kappa \sim \Theta(d^2)$ due to dense matrix storage for H_1 , and $\mu \sim \Theta(d^{\frac{1}{2}})$ (see App. E). The scaling of required memory resources is illustrated in Fig. 2(b): memory usage scales with dimension d as $\Theta(d^2)$ for both full-AD and HG, consistent with $\kappa \sim \Theta(d^2)$ being the dominant contribution. As anticipated, the memory usage scaling with respect to N differs between full-AD and HG [Fig. 2(c)]: for full-AD, the scaling is linear with N ; for HG, it is independent of N . This confirms the favorable memory efficiency of HG compared to full-AD in optimization problems with large number of time steps.

⁵ Another way of calculating ∇C_1^g consists of parallelized forward-backward propagation of all basis states. The concrete scaling behavior then depends on the parallelization scheme; that discussion is beyond the scope of this paper.

⁶ AD is implemented by the package Autograd [38] in this benchmark. This package does not support sparse linear algebra, so we implement dense matrix storage for the Hamiltonian in both full-AD and HG for fair comparison.

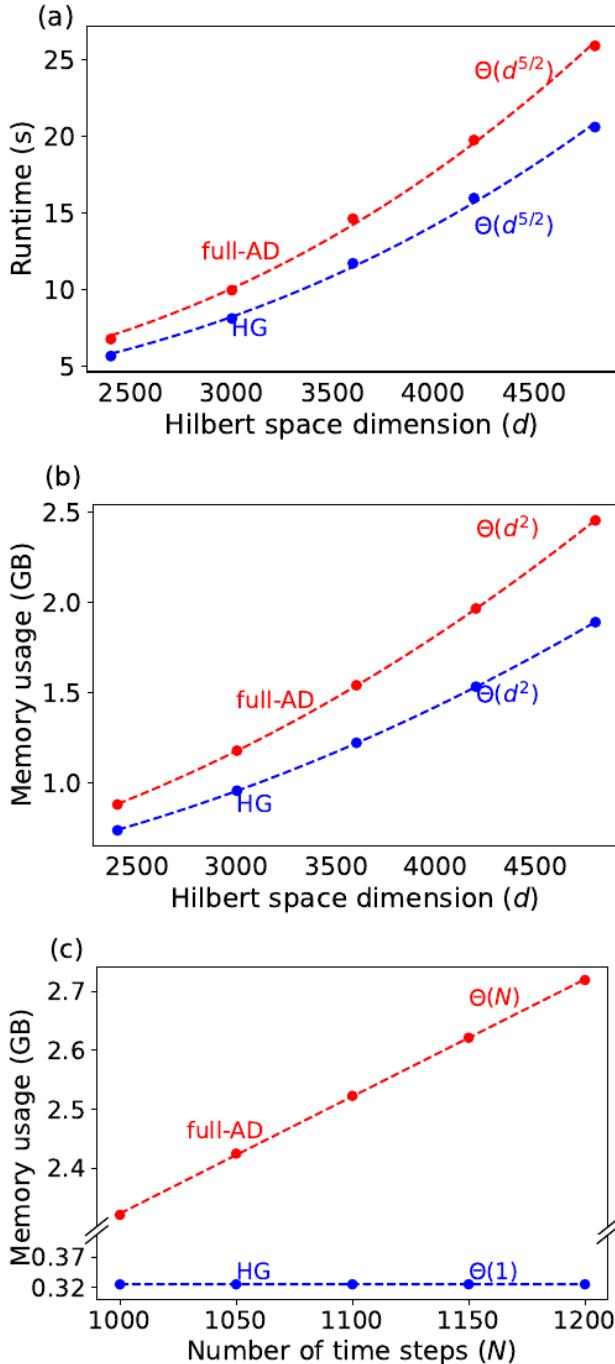


図2. 空洞とトランスマントンが結合した系におけるフォック状態生成のベンチマーク結果。(a) ヒルベルト空間次元 d に対する実行時間と進化時間ステップあたりのメモリ使用量。トランスマントン次元を6に固定し、キャビティ次元を大きくすることで次元を変化させた。すべてのパネルにおいて、データポイントはベンチマークによる結果であり、破線はべき乗則によるフィットである。(結果は以下のパラメータで得られた: $g/2\pi = 0.1 \text{ GHz}$, $\alpha/2\pi = -0.225 \text{ GHz}$, $a^{(v)}/2\pi = 0.1 \text{ GHz}$ の場合に得られる: $\alpha/2\pi = 3 \text{ GHz}$, $g/2\pi = 0.1 \text{ GHz}$, $\alpha/2\pi = -0.225 \text{ GHz}$, 一定制御 $a_z/2\pi = a_x/2\pi = 0.1 \text{ GHz}$ での結果です)。

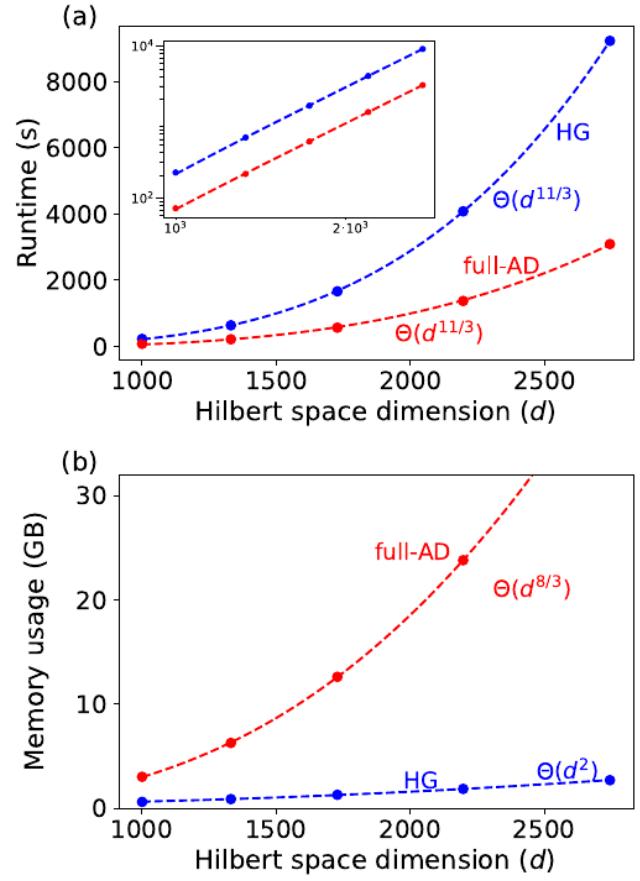


図3. 3つのトランスマントンが結合したシステムにおけるユニタリーゲート最適化のベンチマーク結果。(a) ヒルベルト空間次元 d に対する進化時間ステップあたりのランタイムとメモリ使用量。挿入図は、HGとfull-ADのランタイムスケーリングが同じべき乗則を持つことを確認しています。すべてのパネルにおいて、データポイントはベンチマークによる結果であり、破線はべき乗則によるフィットである。(結果は以下のパラメータで得られた: $g/2\pi = 0.1 \text{ GHz}$, $\alpha/2\pi = -0.225 \text{ GHz}$, $a^{(v)}/2\pi = 0.1 \text{ GHz}$).

次に、ユニタリーゲートを最適化する例について説明します。このベンチマークは、最近接結合を持つ3つの駆動トランスマントンに対して行われる。対象となるゲートは、3つのトランスマントンのそれぞれに対する同時ハダマード演算で構成されています。具体的には、同じ周波数を持つトランスマントンが共振して駆動する場合を考えます。駆動部の回転フレームにおける完全なハミルトニアンは次の通りである。

$$H_2(t) = \sum_{\nu=1}^3 \left[\frac{1}{2} \alpha b_\nu^\dagger b_\nu (b_\nu^\dagger b_\nu - 1) + a^{(\nu)}(t) (b_\nu + b_\nu^\dagger) \right] + g(b_1 b_2^\dagger + b_2 b_3^\dagger + \text{h.c.}) \quad (21)$$

最適化の目標は、不貞度 C_2^g を最小化することである。このベンチマークの設定は、前の例と同じである。図3は、1つの時間ステップ($N = 1$)に基づくベンチマークを記録しています。

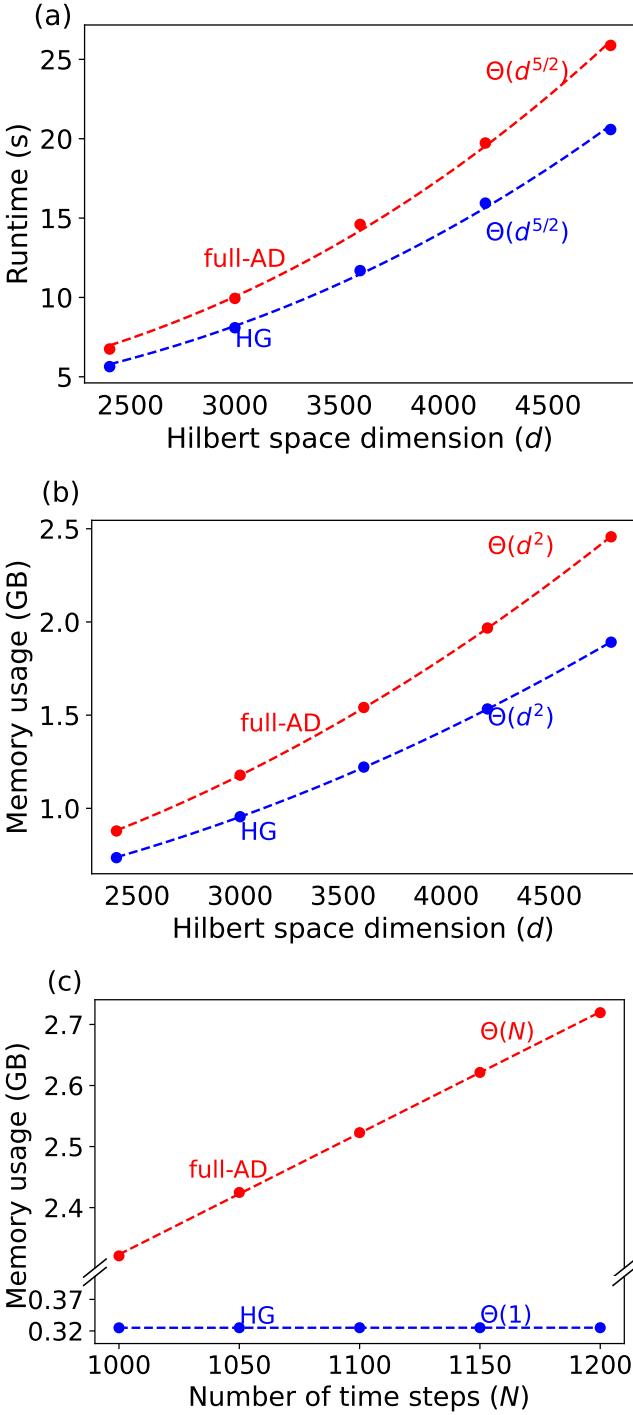


Figure 2. Benchmark results for Fock state generation in a system consisting of a cavity coupled to a transmon. (a) Runtime and (b) memory usage per evolution time step versus Hilbert space dimension d . The dimension is varied by increasing cavity dimension while fixing the transmon dimension to 6. (c) Memory usage as a function of the number of time steps for $d = 600$. In all panels, data points are results from benchmarking and dashed lines are power law fits. (Results are obtained for: $\Delta/2\pi = 3$ GHz, $g/2\pi = 0.1$ GHz, $\alpha/2\pi = -0.225$ GHz and constant controls $a_z/2\pi = a_x/2\pi = 0.1$ GHz.)

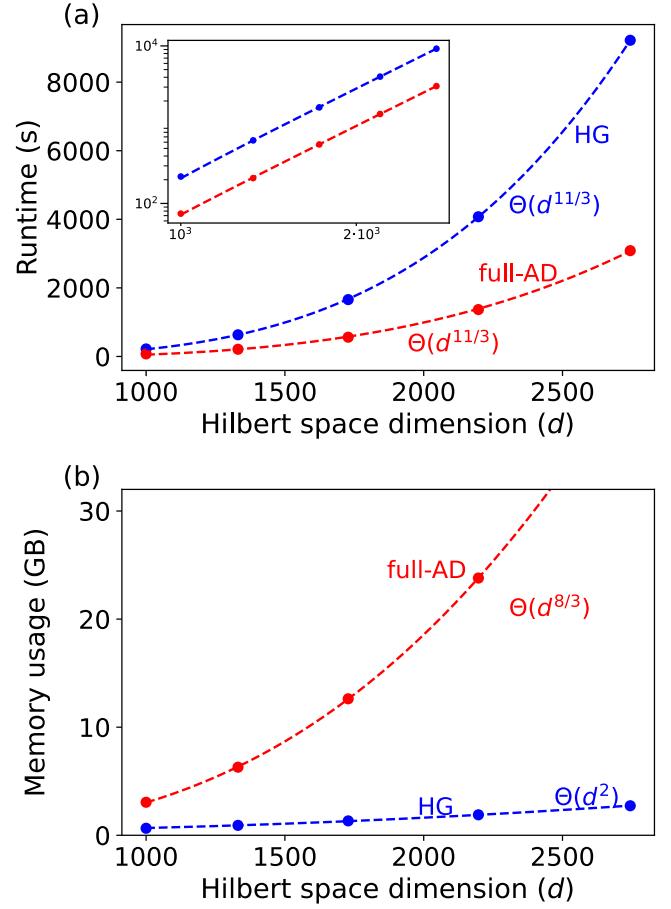


Figure 3. Benchmark results for unitary gate optimization in a system consisting of three coupled transmons. (a) Runtime and (b) memory usage per evolution time step versus Hilbert space dimension d . The dimension is increased by enlarging the dimension of each transmon simultaneously. The inset confirms that runtime scaling for HG and full-AD has the same power law. In all panels, data points are results from benchmarking and dashed lines are power law fits. (Results are obtained with the following parameters: $g/2\pi = 0.1$ GHz, $\alpha/2\pi = -0.225$ GHz and $a^{(\nu)}/2\pi = 0.1$ GHz.)

We next turn to an example for optimizing a unitary gate. This benchmark is performed for three driven transmons with nearest neighbor coupling. The target gate consists of simultaneous Hadamard operations on each of the three transmons. For concreteness, we consider the case of transmons with the same frequency, driven resonantly. The full Hamiltonian in the rotating frame of the drive is

$$H_2(t) = \sum_{\nu=1}^3 \left[\frac{1}{2} \alpha b_\nu^\dagger b_\nu (b_\nu^\dagger b_\nu - 1) + a^{(\nu)}(t)(b_\nu + b_\nu^\dagger) \right] + g(b_1 b_2^\dagger + b_2 b_3^\dagger + \text{h.c.}) \quad (21)$$

The goal of the optimization is to minimize the infidelity C_2^g . The setup for this benchmark is the same as in the previous example.

Fig. 3 records the benchmark based on a single time step

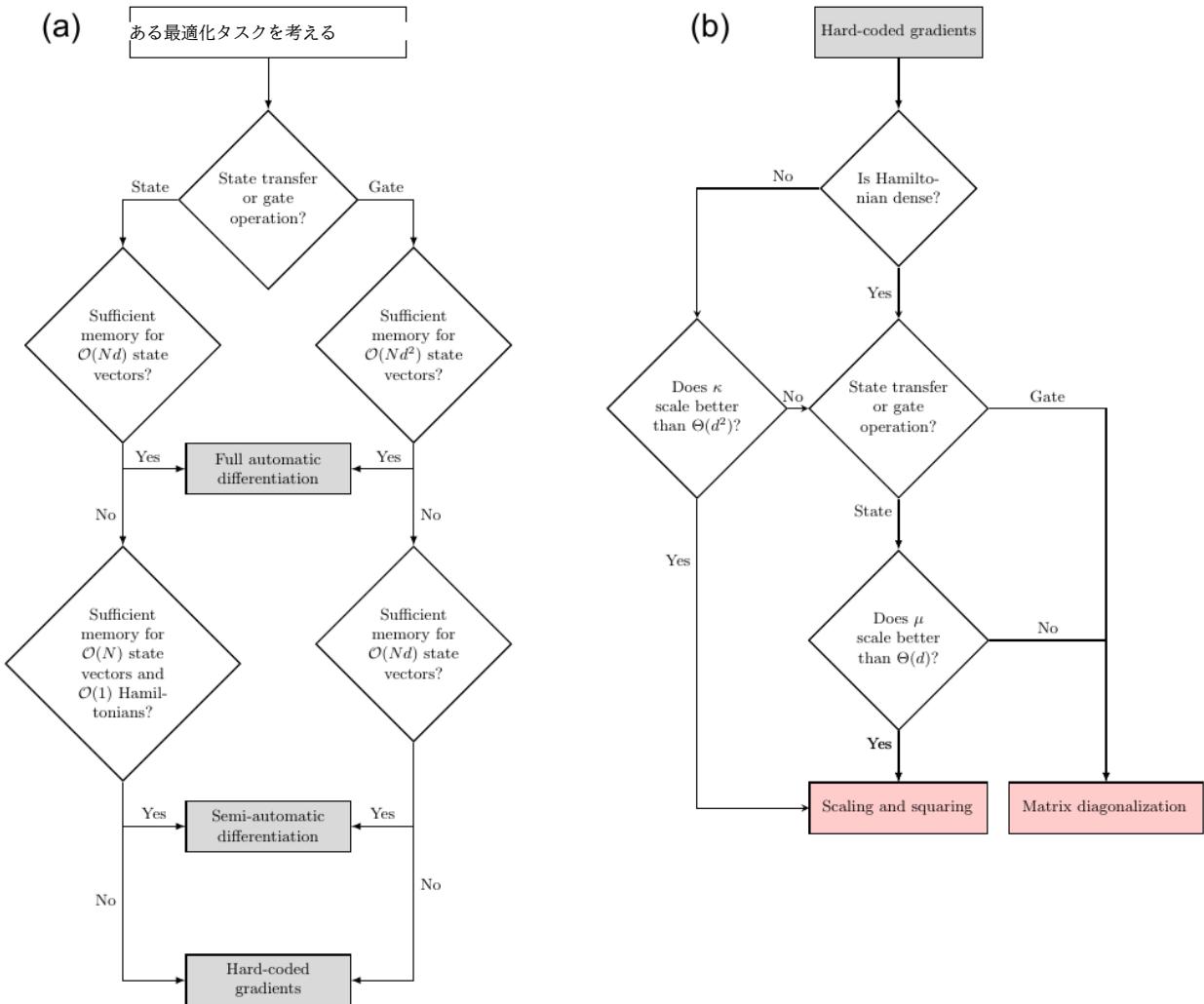


図4. 最適な数値戦略を選択するためのデシジョンツリー。(a) 勾配を評価する3つの数値手法の中から選択する：完全自動微分、半自動微分、ハードコードされた分析的勾配。(ここでは、 $\mu \sim \Theta(d)$ と仮定する。I.) (b) 伝搬体状態積を評価するための適切な戦略を決定する。

次元 d に対する実行時間のスケーリングは、full-ADとHGの両方で $\Theta(d^3)$ であることが観測された（図3(a)参照）。このべき乗則は、 $\kappa \sim \Theta(d^2)$ と $\mu \sim \Theta(d^2)$ の $\Theta(\mu \kappa d)$ から生じる [App. E 参照]。図3(b)に d に対するメモリ使用量のスケーリングを示すが、full-ADはHGと比較して余分な 2μ $\Theta(d^3)$ 依存性があることが確認された (Tab. I)。このため、今回のベンチマークでは、full-ADはHGの約20倍のメモリを消費しています。このことから、Hilbert空間の次元が大きい最適化問題において、メモリ制約が制限となる場合は、HGが望ましいことがわかる。この改善は、実行時のスケーリング⁷を悪化させることなく達成されたことを強調する。

V. CHOOSING AMONG DIFFERENT OPTIMIZATION STRATEGIES

原理的に行列指数化のスケーリング $\Theta(N d^3)$ を超えることができることに注意する必要がある。ある最適化問題に対して、フルAD、HGまたはセミADが最適戦略であるかを決めるいくつかの考慮事項がある。ヒルベルト空間の次元 d と時間ステップ数 N が十分に小さい場合、メモリ使用量がボトルネックにならないことがある。この場合、ハードコードされた勾配が不要になるため、full-ADが望ましいかもしれません。full-ADのメモリコストが手頃でなく、解析的な勾配を計算するのが面倒な場合は、HGよりメモリ効率は劣るもの、semi-ADが有効な妥協案となる可能性があります。それ以外の場合は

⁷ HGのスケーリングと二乗による実行時間スケーリング $\Theta(N \mu \kappa d)$ は、

行列の対角化 (App. F 参照) に基づいて行う。この場合、後者の方が有利です。

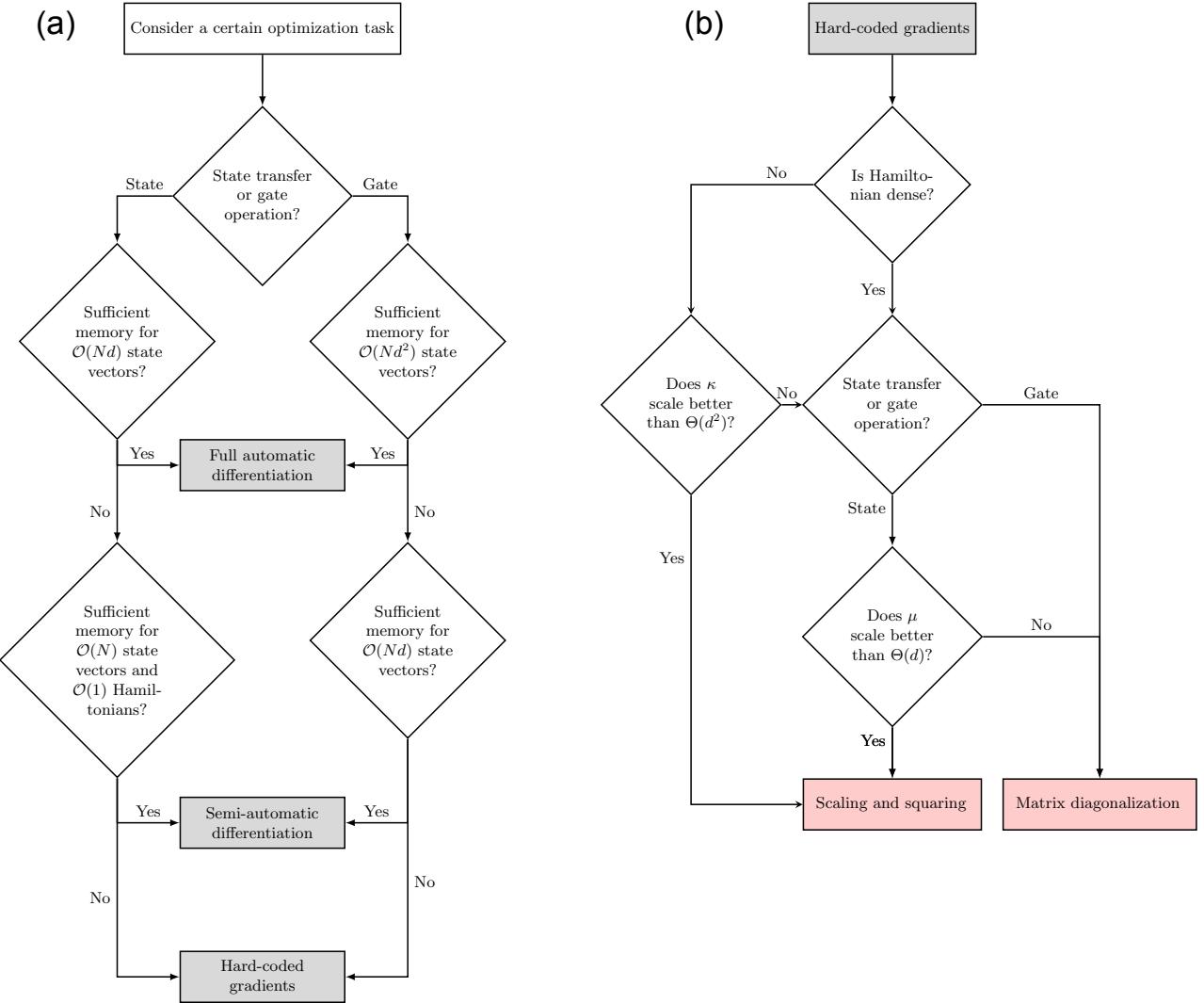


Figure 4. Decision trees for selecting the optimal numerical strategy. (a) Choosing among three numerical methods for evaluating gradients: full automatic differentiation, semi-automatic differentiation, hard-coded analytical gradients. (Here, we assume $\mu \sim \Theta(d)$; for different scaling see Tab. I.) (b) Determining the appropriate strategy for evaluating propagator-state products.

($N = 1$). The observed runtime scaling with dimension d is $\Theta(d^{11/3})$ for both full-AD and HG, see Fig. 3(a). This power law arises from $\Theta(\mu\kappa d)$ for $\kappa \sim \Theta(d^2)$ and $\mu \sim \Theta(d^{2/3})$ [see App. E]. The memory usage scaling with d is illustrated in Fig. 3(b), confirming that full-AD has an unfavorable extra $\mu \sim \Theta(d^{2/3})$ dependence compared to HG (Tab. I). Throughout our benchmarks, this causes full-AD to consume about 20 times more memory than HG. This underlines that HG is preferable whenever memory constraints become a limitation in optimization problems with large Hilbert space dimension. We emphasize that this improvement is achieved without worsening the runtime scaling⁷.

V. CHOOSING AMONG DIFFERENT OPTIMIZATION STRATEGIES

There are several considerations which determine whether full-AD, HG or semi-AD is the optimal strategy for a given optimization problem, see the decision tree Fig. 4. Whenever the Hilbert space dimension d and the number of time steps N are sufficiently small, then memory usage may not be a bottleneck. In this case, full-AD may be preferable, since it eliminates the need for hard-coded gradients. If the memory cost of full-AD is not affordable but analytical gradients are tedious to compute, semi-AD may be a viable compromise, though still less memory efficient than HG. In all other cases,

⁷ We note that the runtime scaling $\Theta(N\mu\kappa d)$ of HG via scaling and squaring can in principle exceed the scaling $O(Nd^3)$ for matrix exponentiation

based on matrix diagonalization (see App. F). In this case, the latter is the better option.

HGは、full-ADやsemi-ADと比較して、Nやdが大きい最適化に対応できるため、選択される手法である。HGをスケーリングと2乗、または行列対角化のどちらで実装するかは、ハミルトニアンがスペースで κ が $\Theta(d^2)$ よりも良くスケールするかで判断する必要があります。後者の場合、スケーリングと二乗が好ましく、全体のメモリ使用量 $\Theta(d + \kappa)$ をもたらし、これは行列対角化のメモリスケーリング $\Theta(d^2)$ よりも優れています。一方、 $\kappa \propto \Theta(d^2)$ のハミルトニアンでは、これら2つの方法はメモリ使用量のスケーリングにおいて同等であり、特定の最適化タスクに対してより実行時間の長い方を選択することができます。(1)状態遷移の場合、実行時間は $\Theta(\mu d^2)$ となり、 μ が $\Theta(d)$ よりもスケールが大きい場合は、スケーリングと二乗が行列対角化により優れています。(2)ゲート最適化では、実行時間は $\Theta(\mu d^3)$ のようにスケールするため、行列対角化が望ましい。

VI. CONCLUSIONS

量子最適制御におけるメモリボトルネック問題に動機づけられ、我々はGRAPEの複数のフレーバー（ハードコード勾配(HG)、完全自動微分(full-AD)、半自動微分(semi-AD)）のメモリ使用率と実行時スケーリングを再検討し比較した。これらの3つの方法は、実行時間のスケーリングでは同等であるが、メモリ使用量のスケーリングで特徴的な違いがある。これらの方法のランキングは、最も効率的なものから最も効率的でないものへと向かって、次のようにになります：HG、semi-AD、full-ADです。したがって、メモリボトルネックの問題に直面した場合、HGが好ましい選択肢となる。我々は、HGとfull-ADのスケーリングを、それぞれ具体的な状態遷移とゲートオペレーションの最適化に関するベンチマークスタディによって説明した。その一環として、プロパゲータ・ステート積を数値で評価するために使用されるスケーリングとスクウェアリングの実装の改良を発表しました。研究した例では、Scipyのexpm乗算関数と比較して高速化が確認されました。図4に示す決定木は、与えられた最適化問題に対する適切な数値戦略の選択をまとめたものである。

ACKNOWLEDGEMENT

Thomas Propson の実りある議論に感謝する。本資料は、米国エネルギー省科学局、国立量子情報科学研究センター、超伝導量子材料・システムセンター(SQMS)の契約番号DE-AC02-07CH11359の支援を受けた研究に基づいています。本研究で使用した以下のオープンソースパッケージに感謝する：matplotlib [46]、numpy [47]、scipy [48]、sympy [49]、scqubits [50]、and qoc [32]。

Appendix A: Analytical gradients

本文では、状態遷移またはゲート不実性を最小化することにのみ焦点を当てた。この付録では、関心のある他のコスト貢献についての解析的勾配を提供する。コスト貢献

$$C_2^{st} = \frac{1}{N} \sum_{n'=1}^N \langle \psi_{n'} | \Omega | \psi_{n'} \rangle \quad (\text{A1})$$

は、エルミート演算子 Ω の積分期待値にペナルティを与える。このコスト寄与の勾配は次のように書くことができる。

$$\begin{aligned} \frac{\partial C_2^{st}}{\partial a_n} &= \frac{1}{N} \sum_{n'=1}^N \langle \psi_{n'} | \Omega \frac{\partial}{\partial a_n} \prod_{n''=1}^{n'} U_{n''} | \psi_0 \rangle + c.c. \\ &= \frac{1}{N} \left(\sum_{n'=n}^N \prod_{n''=n+1}^{n'} \langle \psi_{n'} | \Omega U_{n''} \frac{\partial}{\partial a_n} U_n | \psi_{n-1} \rangle \right) + c.c. \\ &= \frac{2}{N} \operatorname{Re}(\langle \Phi_n | \frac{\partial}{\partial a_n} U_n | \psi_{n-1} \rangle), \end{aligned} \quad (\text{A2})$$

with

$$|\Phi_n\rangle := \sum_{n'=n}^N U_{n+1}^\dagger \cdots U_{n'}^\dagger \Omega |\psi_{n'}\rangle \quad (\text{A3})$$

ベクトル $|\Phi_n\rangle$ は、再帰関係に従っている。

$$|\Phi_n\rangle = U_{n+1}^\dagger |\Phi_{n+1}\rangle + \Omega |\psi_n\rangle. \quad (\text{A4})$$

この関係から、式(A2)の式は、第III章Cで議論したものと同様の前進後退方式で評価することができる。コスト寄与度 $C_3^{st} = \sum_{n=0}^{PN} |\hbar \phi_T | \phi_n |^2$ は、すべての時間ステップからの不忠実度を合計する(C_1^{st} のように最終状態の不忠実度を記録するだけとは対照的に)。これにより、システムをより速く目標状態に導くことができ、状態遷移の時間を短縮することができる。式(A2)と同様に、 C_3^{st} の勾配は次のようになる：

$$\frac{\partial C_3^{st}}{\partial a_n} = -\frac{2}{N} \operatorname{Re}(\langle \Phi_{T,n} | \frac{\partial}{\partial a_n} U_n | \psi_{n-1} \rangle), \quad (\text{A5})$$

with

$$|\Phi_{T,n}\rangle := \sum_{n'=n}^N U_{n+1}^\dagger \cdots U_{n'}^\dagger |\phi_T\rangle \langle \phi_T | \psi_{n'}\rangle. \quad (\text{A6})$$

ベクトルは再帰的関係に従う。

$$|\Phi_{T,n}\rangle = U_{n+1}^\dagger |\Phi_{T,n+1}\rangle + |\phi_T\rangle \langle \phi_T | \psi_n\rangle. \quad (\text{A7})$$

これにより、式(A5)の勾配式を評価するために前方-後方スキームを使用することができます。ゲート演算の場合、 C_3^{st} に類するコスト寄与度は、 $C_3^g = \sum_{n=0}^{PN} \operatorname{tr} U_T^\dagger U_n | \psi_n \rangle \langle \psi_n | / N d^2$ 、

HG is the method of choice, since it can handle optimization with larger N and d compared to full-AD and semi-AD.

The choice of implementing HG via scaling and squaring or matrix diagonalization should be informed by whether the Hamiltonian is sparse and κ scales better than $\Theta(d^2)$. In the latter case, scaling and squaring is preferable and leads to an overall memory usage $\sim \Theta(d + \kappa)$, which is better than the memory scaling $O(d^2)$ of matrix diagonalization. By contrast, for Hamiltonians with $\kappa \sim \Theta(d^2)$, these two methods are equivalent in memory usage scaling and we are free to select the one with better runtime for a specific optimization task. While the runtime for optimization using matrix diagonalization scales as $O(d^3)$ [App. F], the runtime for optimization based on scaling and squaring differs between state-transfer and gate optimizations: (1) for state transfer, the runtime scales as $\Theta(\mu d^2)$; hence, whenever μ scales better than $\Theta(d)$, scaling and squaring wins over matrix diagonalization. (2) for gate optimization, the runtime scales as $\Theta(\mu d^3)$; hence matrix diagonalization is preferable.

VI. CONCLUSIONS

Motivated by the memory bottleneck problem in quantum optimal control of large systems, we have revisited and compared the memory usage and runtime scaling of multiple flavors of GRAPE: hard-coded gradients (HG), full automatic differentiation (full-AD) and semi-automatic differentiation (semi-AD). These three methods are equivalent in runtime scaling but differ characteristically in scaling of memory usage. The ranking among them is as follows, going from the most to least efficient: HG, semi-AD and full-AD. Thus HG is the preferred option when facing a memory bottleneck problem.

We have illustrated the scaling of HG and full-AD by benchmark studies for concrete state-transfer and gate-operation optimizations, respectively. As part of this, we have presented improvements in the implementation of scaling and squaring, used to evaluate propagator-state products numerically. In the example studied, we observed a speedup compared to Scipy's `expm_multiply` function. The decision tree shown in Fig. 4 summarizes the choice of appropriate numerical strategy for a given optimization problem.

ACKNOWLEDGEMENT

We thank Thomas Propson for fruitful discussions. This material is based upon work supported by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Superconducting Quantum Materials and Systems Center (SQMS) under contract number DE-AC02-07CH11359. We are grateful for the following open-source packages used in our work: `matplotlib` [46], `numpy` [47], `scipy` [48], `sympy` [49], `scqubits` [50], and `qoc` [32].

Appendix A: Analytical gradients

In the main text, we exclusively focused on minimizing state-transfer or gate infidelity. In this appendix, we provide the analytical gradients for other cost contributions of interest.

The cost contribution

$$C_2^{st} = \frac{1}{N} \sum_{n'=1}^N \langle \psi_{n'} | \Omega | \psi_{n'} \rangle \quad (\text{A1})$$

penalizes the integrated expectation value of the Hermitian operator Ω . The gradient of this cost contribution can be written as

$$\begin{aligned} \frac{\partial C_2^{st}}{\partial a_n} &= \frac{1}{N} \sum_{n'=1}^N \langle \psi_{n'} | \Omega \frac{\partial}{\partial a_n} \prod_{n''=1}^{n'} U_{n''} | \psi_{n'} \rangle + c.c. \\ &= \frac{1}{N} \left(\sum_{n'=n}^N \prod_{n''=n+1}^{n'} \langle \psi_{n'} | \Omega U_{n''} \frac{\partial}{\partial a_n} U_n | \psi_{n-1} \rangle \right) + c.c. \\ &= \frac{2}{N} \operatorname{Re}(\langle \Phi_n | \frac{\partial}{\partial a_n} U_n | \psi_{n-1} \rangle), \end{aligned} \quad (\text{A2})$$

with

$$|\Phi_n\rangle := \sum_{n'=n}^N U_{n+1}^\dagger \cdots U_{n'}^\dagger \Omega | \psi_{n'} \rangle \quad (\text{A3})$$

The vector $|\Phi_n\rangle$ obeys the recursion relation

$$|\Phi_n\rangle = U_{n+1}^\dagger |\Phi_{n+1}\rangle + \Omega |\psi_n\rangle. \quad (\text{A4})$$

Due to this relation, the expression in Eq. (A2) can be evaluated with a forward-backward scheme analogous to the one discussed in the Sec. III C.

The cost contribution $C_3^{st} = 1 - \frac{1}{N} \sum_{n'=1}^N |\langle \phi_T | \psi_{n'} \rangle|^2$ sums up infidelities from all time steps (as opposed to recording the final state infidelity only, as in C_1^{st}). This steers the system towards its target state more rapidly, thus helping to reduce the duration of state-transfer. Similar to Eq. (A2), the gradient of C_3^{st} is:

$$\frac{\partial C_3^{st}}{\partial a_n} = -\frac{2}{N} \operatorname{Re}(\langle \Phi_{T,n} | \frac{\partial}{\partial a_n} U_n | \psi_{n-1} \rangle), \quad (\text{A5})$$

with

$$|\Phi_{T,n}\rangle := \sum_{n'=n}^N U_{n+1}^\dagger \cdots U_{n'}^\dagger |\phi_T\rangle \langle \phi_T | \psi_{n'} \rangle. \quad (\text{A6})$$

The vector obeys the recursive relation

$$|\Phi_{T,n}\rangle = U_{n+1}^\dagger |\Phi_{T,n+1}\rangle + |\phi_T\rangle \langle \phi_T | \psi_n \rangle. \quad (\text{A7})$$

This allows us to use a forward-backward scheme to evaluate the gradient expression in Eq. (A5).

For gate operations, the cost contribution analogous to C_3^{st} is given by $C_3^g = 1 - \sum_{n'=1}^N |\operatorname{tr}(U_T^\dagger U_{n',1})|^2 / Nd^2$, where

ここで $U_n|0\rangle = U_n|0\rangle \dots U_1\rangle$ は時間ステップ1から n^0 までの進化を記述している。このとき、勾配は

$$\frac{\partial C_3^g}{\partial a_n} = -\frac{2}{Nd^2} \sum_{h=1}^d \operatorname{Re} \left(\langle \Phi_{T,n}^h | \frac{\partial}{\partial a_n} U_n | \psi_{n-1}^h \rangle \right), \quad (\text{A8})$$

with

$$|\Phi_{T,n}^h\rangle := \sum_{n'=n}^N U_{n+1}^\dagger \cdots U_{n'}^\dagger U_T |\psi_0^h\rangle \operatorname{tr}(U_{n',1} U_T^\dagger). \quad (\text{A9})$$

ここで、 $\{\psi_i^h\}$ は任意の直交基底を形成し、 $h = 1, 2, \dots, d$ は基底状態を列挙する。ベクトル $|\Phi_{T,n}^h\rangle$ は次の再帰関係に従う。

$$|\Phi_{T,n}^h\rangle = U_{n+1}^\dagger |\Phi_{T,n+1}^h\rangle + U_T |\psi_0^h\rangle \operatorname{tr}(U_{n,1} U_T^\dagger), \quad (\text{A10})$$

また、前方-後方伝搬スキームで勾配式を評価することができるようになりました。

Appendix B: Scaling analysis

App. Aで議論したコスト貢献について、次にfull-AD、semi-AD、HGのメモリ使用量と実行時のスケーリングを分析する。
a. スケーリング分析：ハードコードされた勾配。式の類似性と再帰関係の存在により、図1のような前方-後方スキームは、 ∇C_2^{st} 、 ∇C_3^{st} 、 ∇C_3^g に対しても適用可能である。その結果、第IV A章のスケーリング解析は、 ∇C_1^{st} と ∇C_2^g と同じメモリ使用量と実行時間をもたらすことがわかった。
b. スケーリング解析：全自動微分。 C_2^{st} 、 C_3^{st} 、 C_3^g というコスト貢献度の表現に違いがあるものの、ランタイムとメモリ使用量のスケーリングは、中間ベクトルの保存と行列とベクトルの乗算を繰り返す必要性によって再び決まっていることがわかります。その結果、スケーリングは、第IV A章で議論したコスト貢献と同じままである。要約すると、状態遷移とゲート操作のコスト貢献のスケーリングは、表Iに示されたものと一致する。semi-ADのスケーリングについては、Ref. 34。

付録C：上限値 $E_A(m, s)$ を決定する。

第III章Bで $e^A|\Psi\rangle$ を近似するためには、誤差 $\varepsilon_{A,\Psi_i}(m, s)$ の上界 $E_A(m, s)$ [Eq.(14)]を決定することが必要でした。この付録では、そのような境界を得る方法を示す。誤差の式[式(14)]を考えると、分母が単に1であることに注意する。誤差 ε_{A,Ψ_i} は次のように境界を定められる：

$$\begin{aligned} \varepsilon_{A,\Psi} &= \left\| e^A |\Psi\rangle - (e_m^B)^s |\Psi\rangle + (e_m^B)^s |\Psi\rangle - \llbracket (e_m^B)^s |\Psi\rangle \rrbracket \right\|_2 \\ &\leq \underbrace{\left\| e^A |\Psi\rangle - (e_m^B)^s |\Psi\rangle \right\|_2}_{\varepsilon_t} + \underbrace{\left\| (e_m^B)^s |\Psi\rangle - \llbracket (e_m^B)^s |\Psi\rangle \rrbracket \right\|_2}_{\varepsilon_r}. \end{aligned} \quad (\text{C1})$$

誤差 ε_t はテイラー系列の切り捨てに起因し、丸め誤差 ε_r は有限精度の算術に起因する。この2つの誤差の上界を別々に導出することにする。

1. 切り捨て誤差 ε_t の上界

切り捨て誤差を次のように書き換えます。

$$\begin{aligned} \varepsilon_t &= \left\| e^A |\Psi\rangle - (e^B - R_m(B))^s |\Psi\rangle \right\|_2 \\ &= \left\| \sum_{i=1}^s \frac{s!}{i!(s-i)!} (-R_m(B))^i (e^B)^{s-i} |\Psi\rangle \right\|_2, \end{aligned} \quad (\text{C2})$$

ここで、 $R_m(B) = \sum_{q=m+1}^{\infty} B^q / q!$ は、切り捨てられた指數級数の行列値誤差を表す。三角形の不等式とsubmultiplicativity [51]を用いると、以下の境界が得られる。

$$\begin{aligned} \varepsilon_t &\leq \sum_{i=1}^s \frac{s!}{i!(s-i)!} (R_m(\|B\|_2))^i \|e^B\|_2^{s-i} \|\Psi\|_2 \\ &= \sum_{i=1}^s \frac{s!}{i!(s-i)!} (R_m(\|B\|_2))^i. \end{aligned} \quad (\text{C3})$$

ここで2行目は、状態が正規化され、 e^B がユニタリーであることを利用する。この上界を数値的に評価して m と s を決定する必要があるが、行列2-normの計算は固有値解法が必要であり不便である。これを軽減するために、⁸を利用して行列1-normに切り替えます。

$$\|B\|_2 \leq \|B\|_1. \quad (\text{C4})$$

Monotonicity of R_m leads us to

$$\begin{aligned} \varepsilon_t &\leq \sum_{i=1}^s \frac{s!}{i!(s-i)!} (R_m(\|B\|_1))^i \\ &< s R_m(\|B\|_1) \frac{1 - (s R_m(\|B\|_1))^s}{1 - s R_m(\|B\|_1)}. \end{aligned} \quad (\text{C5})$$

2. 丸め誤差 ε_r の上界

丸め誤差 ε_r は、実数の浮動小数点表現の精度が有限であることと、基本的な算術演算の精度が有限であることに起因している。実数 δ の浮動小数点表現は、 $\delta' = \delta/\epsilon$ が相対偏差であると書ける[51]。その大きさは常に機械精度より小さい、すなわち $|\delta'| < u$ 。同様に、複素数 z は、次のように従います。

$$\llbracket z \rrbracket = z(1 + \delta) \quad (\text{C6})$$

⁸ Generally, $\|B\|_2 \leq \sqrt{\|B\|_\infty \|B\|_1}$ [52] holds true for any B . When B is anti-Hermitian, $\|B\|_\infty = \|B\|_1$ leads to inequality Eq. (C4).

$U_{n',1} = U_{n'} \cdots U_1$ describes the evolution from time step 1 to n' . We obtain the gradient

$$\frac{\partial C_3^g}{\partial a_n} = -\frac{2}{Nd^2} \sum_{h=1}^d \operatorname{Re} \left(\langle \Phi_{T,n}^h | \frac{\partial}{\partial a_n} U_n | \psi_{n-1}^h \rangle \right), \quad (\text{A8})$$

with

$$|\Phi_{T,n}^h\rangle := \sum_{n'=n}^N U_{n+1}^\dagger \cdots U_{n'}^\dagger U_T |\psi_0^h\rangle \operatorname{tr}(U_{n',1} U_T^\dagger). \quad (\text{A9})$$

Here, $\{|\psi_0^h\rangle\}$ forms an arbitrary orthonormal basis and $h = 1, 2, \dots, d$ enumerates the basis states. The vector $|\Phi_{T,n}^h\rangle$ obeys the recursion relation

$$|\Phi_{T,n}^h\rangle = U_{n+1}^\dagger |\Phi_{T,n+1}^h\rangle + U_T |\psi_0^h\rangle \operatorname{tr}(U_{n,1} U_T^\dagger), \quad (\text{A10})$$

again enabling the evaluation of the gradient expression with a forward-backward propagation scheme.

Appendix B: Scaling analysis

For the cost contributions discussed in App. A, we now analyze the memory usage and runtime scaling of full-AD, semi-AD and HG.

a. *Scaling analysis: hard-coded gradient.* Thanks to the similarities in the expressions and the existence of recursion relations, forward-backward schemes akin to Fig. 1 can also be applied to ∇C_2^{st} , ∇C_3^{st} and ∇C_3^g . As a result, we find that the scaling analysis in Sec. IV A leads to the same the memory usage and runtime requirement as ∇C_1^{st} and ∇C_2^{st} .

b. *Scaling analysis: full-automatic differentiation.* Despite the specific differences among the expressions of cost contributions C_2^{st} , C_3^{st} and C_3^g , we find that the scaling of runtime and memory usage is again set by the need to store intermediate vectors and repeatedly perform matrix-vector multiplications. Consequently, the scaling remains the same as for the cost contributions discussed in Sec. IV A.

In summary, the scaling for the state-transfer and gate-operation cost contributions matches those shown in table I. For the scaling of semi-AD, see Ref. 34.

Appendix C: Determine the upper bound $E_A(m, s)$

The approximation of $e^A|\Psi\rangle$ in Sec. III B required the determination of an upper bound $E_A(m, s)$ for the error $\varepsilon_{A,|\Psi\rangle}(m, s)$ [Eq. (14)]. In this appendix, we show how to acquire such bound.

Considering the expression for the error [Eq. (14)], we note that the denominator is simply 1. The error $\varepsilon_{A,|\Psi\rangle}$ is bounded as follows:

$$\begin{aligned} \varepsilon_{A,|\Psi\rangle} &= \left\| e^A |\Psi\rangle - (e_m^B)^s |\Psi\rangle + (e_m^B)^s |\Psi\rangle - \llbracket (e_m^B)^s |\Psi\rangle \rrbracket \right\|_2 \\ &\leq \underbrace{\left\| e^A |\Psi\rangle - (e_m^B)^s |\Psi\rangle \right\|_2}_{\varepsilon_t} + \underbrace{\left\| (e_m^B)^s |\Psi\rangle - \llbracket (e_m^B)^s |\Psi\rangle \rrbracket \right\|_2}_{\varepsilon_r}. \end{aligned} \quad (\text{C1})$$

The error ε_t arises from the truncation of the Taylor series and rounding error ε_r is due to finite-precision arithmetics. We proceed to derive the upper bounds for these two errors separately.

1. Upper bound for the truncation error ε_t

We rewrite the truncation error as

$$\begin{aligned} \varepsilon_t &= \left\| e^A |\Psi\rangle - (e^B - R_m(B))^s |\Psi\rangle \right\|_2 \\ &= \left\| \sum_{i=1}^s \frac{s!}{i!(s-i)!} (-R_m(B))^i (e^B)^{s-i} |\Psi\rangle \right\|_2, \end{aligned} \quad (\text{C2})$$

where $R_m(B) = \sum_{q=m+1}^{\infty} B^q/q!$ denotes the matrix-valued error of the truncated exponential series. Employing the triangle inequality and submultiplicativity [51], we find the bound

$$\begin{aligned} \varepsilon_t &\leq \sum_{i=1}^s \frac{s!}{i!(s-i)!} (R_m(\|B\|_2))^i \|e^B\|_2^{s-i} \|\Psi\|_2 \\ &= \sum_{i=1}^s \frac{s!}{i!(s-i)!} (R_m(\|B\|_2))^i. \end{aligned} \quad (\text{C3})$$

Here the second line utilizes the fact that the state is normalized and e^B is unitary.

This upper bound must be evaluated numerically to determine m and s . However, computing the matrix 2-norm is inconvenient since it requires the use of an eigensolver. To mitigate this, we switch to the matrix 1-norm by exploiting⁸

$$\|B\|_2 \leq \|B\|_1. \quad (\text{C4})$$

Monotonicity of R_m leads us to

$$\begin{aligned} \varepsilon_t &\leq \sum_{i=1}^s \frac{s!}{i!(s-i)!} (R_m(\|B\|_1))^i \\ &< s R_m(\|B\|_1) \frac{1 - (s R_m(\|B\|_1))^s}{1 - s R_m(\|B\|_1)}. \end{aligned} \quad (\text{C5})$$

2. Upper bound for the rounding error ε_r

The rounding error ε_r originates from the finite precision of the floating-point representation of real numbers as well as the finite accuracy of basic arithmetic operations. The floating-point representation $\llbracket \xi \rrbracket$ for the real number ξ can be written as $\llbracket \xi \rrbracket = \xi(1 + \delta)$ where $\delta = \delta(\xi)$ is the relative deviation [51]. Its magnitude is always smaller than machine precision, i.e. $|\delta| < u$. Similarly, the complex number z obeys

$$\llbracket z \rrbracket = z(1 + \delta) \quad (\text{C6})$$

⁸ Generally, $\|B\|_2 \leq \sqrt{\|B\|_{\infty} \|B\|_1}$ [52] holds true for any B . When B is anti-Hermitian, $\|B\|_{\infty} = \|B\|_1$ leads to inequality Eq. (C4).

とし、 $|\delta| \leq u$ とする。2つの複素数 z_1, z_2 による加算や乗算などの算術演算は、次のような相対偏差 $\delta = \delta(z_1, z_2)$ を生じる。

$$\begin{aligned} \|z_1 + z_2\| &= (\|z_1\| + \|z_2\|)(1 + \delta), |\delta| \leq u, \\ \|z_1 z_2\| &= \|z_1\| \|z_2\| (1 + \delta), |\delta| \leq u', \end{aligned} \quad (\text{C7})$$

ここで、 $u^0 := 2\|z\|/(1 - 2u)$ [51]である。式(C6), (C7)で示される相対的な偏差は、一般に、関係する数および問題のある操作に依存します。以下では、異なる δ を ν という添え字で区別する。式 (C6)、(C7)を用いて、式 (13)に従って $e_m^B |\Psi\rangle$ を評価する際の基本演算であるドット積の評価における丸め誤差の上限を導出します。

a. ドットプロダクトの評価に伴う丸め誤差の上界

'ドットプロデュースを評価する際に発生する丸め誤差。
net is given by $|\Sigma_i - \|\Sigma_i\||$ with $\Sigma_i = \sum_{j=1}^d w_j z_j$, $w_j \in \mathbb{C}^{d_1}$
ここで、 w^T は $B^{Kd} = i\delta T/\sqrt{s}$ の行である。 $d=1, 2$ について
丸め誤差を抑える方法を説明し、その後に一般的な結果を与える。式(C6)と式(C7)に基づいて、次のようになる。

$$\begin{aligned} \|\Sigma_1\| - \Sigma_1 &= \|\|w_1 z_1\| - w_1 z_1\| \\ &< \|w_1\| |z_1| \prod_{\nu=1}^3 (1 + \delta_\nu) - 1 | < \gamma_3 \|w_1\| |z_1|. \end{aligned} \quad (\text{C8})$$

In the last step, we have used $|\prod_{\nu=1}^3 (1 + \delta_\nu) - 1| < \frac{\nu u'}{1 - \nu u'}$ [51], and defined $\gamma_n := \frac{\nu u'}{1 - \nu u'}$. For $d = 2$, the upper bound of the error is

$$\begin{aligned} \|\Sigma_2\| - \Sigma_2 &= |(\|w_1\| + \|w_2 z_2\|)(1 + \delta_1) - \Sigma_2| \\ &< \gamma_4 \sum_{i=1}^2 \|w_i\| |z_i|. \end{aligned} \quad (\text{C9})$$

任意の d に対する一般的な上界は次式で与えられる：

$$\|\Sigma_d\| - \Sigma_d < \gamma_{d+2} \sum_{j=1}^d \|w_j\| |z_j|. \quad (\text{C10})$$

w^T はハミルトニアン行列の1行に比例するので、 H のスパース性は w^T のスパース性を意味する。 w^T のエンタリーが消えることは丸め誤差が発生しないので特別である。従って、より強い上限を得ることができる

$$\|\Sigma_d\| - \Sigma_d < \gamma_{\sigma+2} \sum_{j=1}^d \|w_j\| |z_j| \quad (\text{C11})$$

ここで、 σ は w^T の非ゼロ要素の数である。

b. $e_m^B |\Psi\rangle$ の丸め誤差の上界。 $|\Psi\rangle$

ベクトル $e_m^B |\Psi\rangle$ は、以下の方法で再帰的に評価されます。

$$b_j = \frac{B}{j} b_{j-1}, \quad e_j^B b_0 = b_{j-1} + b_j \quad (\text{C12})$$

ここで、 $j = 1, 2, \dots, m$ 、 b_0 は直交基底の選択下での $|\Psi\rangle$ のベクトル表現である。を評価する前に
we upper bound for the error $\|e_m^B b_0\| - e_m^B b_0\|_2$, we first
ベクトル成分 K の誤差 $|e_j^B b_0|$ の上界を導く。
 $\|e_m^B b_0\| - |e_m^B b_0|$ |. we illustrate how to bound this error
for $m = 0, 1$, and then give the general result.

$m = 0$ の場合、次のような境界線が得られます。

$$|\|e_0^B b_0\|^{(i)} - (e_0^B b_0)^{(i)}| = \|\|b_0\|^{(i)} - b_0^{(i)}\| < \gamma_2 |b_0^{(i)}|. \quad (\text{C13})$$

次に、 $m=1$ の例について、誤差の上限を求める：

$$\begin{aligned} &|\|e_1^B b_0\|^{(i)} - (e_1^B b_0)^{(i)}| \\ &= |(\|b_0\|^{(i)} + \|b_1\|^{(i)})(1 + \delta_1) - (e_1^B b_0)^{(i)}| \\ &= \|\|b_0\|^{(i)}(1 + \delta_1) + \frac{\|Ab_0\|^{(i)}}{s} \prod_{\nu=1}^3 (1 + \delta_\nu)^3 - (e_1^B b_0)^{(i)}\| \\ &\stackrel{(\text{C11})}{<} \gamma_3 |b_0^{(i)}| + \gamma_{\sigma+5} \sum_l |B^{(il)}| |b_0^{(l)}|, \end{aligned} \quad (\text{C14})$$

ここで、 σ_i は行列 B の i 行目の非ゼロ要素の数である。

ここで、3行目は複素数と実数の除算誤差、すなわち $= z / \xi$ ($1 + \delta$)、 $|\delta| < u$ [51] に起因する。任意の m に対する一般的な上界 $\|e_m^B b_0\| - |e_m^B b_0|$ は、次式で与えられる：

$$|\|e_m^B b_0\|^{(i)} - (e_m^B b_0)^{(i)}| < \sum_{k=0}^m \gamma_{k(\sigma_i+2)+m+2} \left(\frac{|B|^k}{k!} |b_0| \right)^i, \quad (\text{C15})$$

ここで、 $|B|$ 、 $|b_0|$ はそれぞれ、要素 $|B^{il}|$ 、 $|b_0^{il}|$ を持つ行列とベクトルを示す。を定義する。

$$\sigma' := \max_i \sigma_i \quad (\text{C16})$$

and using Eq. (C15), we obtain

$$\begin{aligned} &\|\|e_m^B b_0\| - (e_m^B b_0)\|_2 = \|\|e_m^B b_0\| - (e_m^B b_0)\|_2 \\ &< \sum_{k=0}^m \gamma_{k(\sigma'+2)+m+2} \left(\frac{|B|^k}{k!} |b_0| \right)_2 \\ &< \sum_{k=0}^m \gamma_{k(\sigma'+2)+m+2} \frac{\|B\|_2^k}{k!} \|b_0\|_2 \\ &\stackrel{(\text{C4})}{<} \sum_{k=0}^m \gamma_{k(\sigma'+2)+m+2} \frac{\|B\|_1^k}{k!} \|b_0\|_2 \\ &= \beta \|b_0\|_2 \end{aligned} \quad (\text{C17})$$

with $\beta := \sum_{k=0}^m \gamma_{k(\sigma'+2)+m+2} \frac{\|B\|_1^k}{k!}$. Here, $\beta \|b_0\|_2$ the desired upper bound for the rounding error of $e_m^B |\Psi\rangle$.

with $|\delta| \leq u$. Arithmetic operations such as addition and multiplication with two complex numbers z_1, z_2 incur a relative deviation $\delta = \delta(z_1, z_2)$ such that

$$\begin{aligned} \llbracket z_1 + z_2 \rrbracket &= (\llbracket z_1 \rrbracket + \llbracket z_2 \rrbracket)(1 + \delta), |\delta| \leq u, \\ \llbracket z_1 z_2 \rrbracket &= \llbracket z_1 \rrbracket \llbracket z_2 \rrbracket (1 + \delta), |\delta| \leq u', \end{aligned} \quad (\text{C7})$$

where $u' := 2\sqrt{2}u/(1 - 2u)$ [51]. The relative deviation shown in Eqs. (C6), (C7) generally depend on both numbers involved as well as the operation in question. In the following, we will distinguish different δ with subscript ν .

Using the Eqs. (C6), (C7), we derive an upper bound for the rounding error in the evaluation of the dot product which is the basic arithmetic operation in evaluating $e_m^B |\Psi\rangle$ according to Eq. (13).

a. *Upper bound for the rounding error associated with evaluating the dot product*

The rounding error incurred when evaluating the dot product is given by $|\Sigma_d - \llbracket \Sigma_d \rrbracket|$ with $\Sigma_d = \mathbf{w}^T \mathbf{z}$ ($\mathbf{w}, \mathbf{z} \in \mathbb{C}^d$). Here, \mathbf{w}^T is a row of $B = i\delta t H/s$. We illustrate how to bound the rounding error for $d = 1, 2$, and give the general result subsequently. Based on Eqs. (C6) and (C7), we have

$$\begin{aligned} |\llbracket \Sigma_1 \rrbracket - \Sigma_1| &= |\llbracket w_1 z_1 \rrbracket - w_1 z_1| \\ &< |w_1| |z_1| \left| \prod_{\nu=1}^3 (1 + \delta_\nu) - 1 \right| < \gamma_3 |w_1| |z_1|. \end{aligned} \quad (\text{C8})$$

In the last step, we have used $|\prod_{\nu=1}^n (1 + \delta_\nu) - 1| < \frac{nu'}{1-nu'}$ [51], and defined $\gamma_n := \frac{nu'}{1-nu'}$. For $d = 2$, the upper bound of the error is

$$\begin{aligned} |\llbracket \Sigma_2 \rrbracket - \Sigma_2| &= |(\llbracket \Sigma_1 \rrbracket + \llbracket w_2 z_2 \rrbracket)(1 + \delta_1) - \Sigma_2| \\ &< \gamma_4 \sum_{i=1}^2 |w_i| |z_i|. \end{aligned} \quad (\text{C9})$$

The general upper bound for arbitrary d is given by:

$$|\llbracket \Sigma_d \rrbracket - \Sigma_d| < \gamma_{d+2} \sum_{j=1}^d |w_j| |z_j|. \quad (\text{C10})$$

Since \mathbf{w}^T is proportional to a row of the Hamiltonian matrix, sparsity of H implies sparsity of \mathbf{w}^T . Vanishing entries in \mathbf{w}^T is special because they do not incur rounding errors. Thus, one can obtain a stronger upper bound

$$|\llbracket \Sigma_d \rrbracket - \Sigma_d| < \gamma_{\sigma+2} \sum_{j=1}^d |w_j| |z_j| \quad (\text{C11})$$

where σ is the number of non-zero elements in \mathbf{w}^T .

b. *Upper bound for the rounding error of $e_m^B |\Psi\rangle$*

The vector $e_m^B |\Psi\rangle$ is evaluated recursively via

$$b_j = \frac{B}{j} b_{j-1}, \quad e_j^B b_0 = b_{j-1} + b_j \quad (\text{C12})$$

where $j = 1, 2, \dots, m$ and b_0 is a vector representation of $|\Psi\rangle$ under a choice of orthonormal basis. Before evaluating the upper bound for the error $\|\llbracket e_m^B b_0 \rrbracket - e_m^B b_0\|_2$, we first derive an upper bound for the error of an vector component $|\llbracket e_m^B b_0 \rrbracket^{(i)} - (e_m^B b_0)^{(i)}|$. We illustrate how to bound this error for $m = 0, 1$, and then give the general result.

For the case $m = 0$, we have the bound

$$|\llbracket e_0^B b_0 \rrbracket^{(i)} - (e_0^B b_0)^{(i)}| = |\llbracket b_0 \rrbracket^{(i)} - b_0^{(i)}| < \gamma_2 |b_0^{(i)}|. \quad (\text{C13})$$

We then obtain the upper bound of the error for the example of $m = 1$:

$$\begin{aligned} &|\llbracket e_1^B b_0 \rrbracket^{(i)} - (e_1^B b_0)^{(i)}| \\ &= |(\llbracket b_0 \rrbracket^{(i)} + \llbracket b_1 \rrbracket^{(i)})(1 + \delta_1) - (e_1^B b_0)^{(i)}| \\ &= |\llbracket b_0 \rrbracket^{(i)}(1 + \delta_1) + \frac{\llbracket Ab_0 \rrbracket^{(i)}}{s} \prod_{\nu=1}^3 (1 + \delta_\nu)^3 - (e_1^B b_0)^{(i)}| \\ &\stackrel{(\text{C11})}{<} \gamma_3 |b_0^{(i)}| + \gamma_{\sigma_i+5} \sum_l |B^{(il)}| |b_0^{(l)}|, \end{aligned} \quad (\text{C14})$$

where σ_i is number of non-zero elements in i th row of the matrix B . Here, the third line arises from the division error between a complex number and a real number, i.e. $\llbracket z/\xi \rrbracket = z/\xi(1 + \delta)$, $|\delta| < u$ [51]. The general upper bound for arbitrary m is given by:

$$|\llbracket e_m^B b_0 \rrbracket^{(i)} - (e_m^B b_0)^{(i)}| < \sum_{k=0}^m \gamma_{k(\sigma_i+2)+m+2} \left(\frac{|B|^k}{k!} |b_0| \right)^i, \quad (\text{C15})$$

where $|B|, |b_0|$ denote the matrix and the vector with elements $|B^{il}|, |b_0^i|$, respectively. Defining

$$\sigma' := \max_i \sigma_i \quad (\text{C16})$$

and using Eq. (C15), we obtain

$$\begin{aligned} \|\llbracket e_m^B b_0 \rrbracket - (e_m^B b_0)\|_2 &= \|\llbracket e_m^B b_0 \rrbracket - (e_m^B b_0)\|_2 \\ &< \sum_{k=0}^m \gamma_{k(\sigma'+2)+m+2} \left(\frac{|B|^k}{k!} |b_0| \right)_2 \\ &< \sum_{k=0}^m \gamma_{k(\sigma'+2)+m+2} \frac{\|B\|_2^k}{k!} \|b_0\|_2 \\ &\stackrel{(\text{C4})}{<} \sum_{k=0}^m \gamma_{k(\sigma'+2)+m+2} \frac{\|B\|_1^k}{k!} \|b_0\|_2 \\ &= \beta \|b_0\|_2 \end{aligned} \quad (\text{C17})$$

with $\beta := \sum_{k=0}^m \gamma_{k(\sigma'+2)+m+2} \frac{\|B\|_1^k}{k!}$. Here, $\beta \|b_0\|_2$ the desired upper bound for the rounding error of $e_m^B |\Psi\rangle$.

c. Upper bound for the error ε_r

ベクトル $(e_m^B)^s b_0$ は再帰的な関係で評価されます。

$$c_l = e_m^B c_{l-1}, \quad l = 1, 2, \dots, s, \quad (\text{C18})$$

ここで、 $c_0 = b_0$ 。誤差 $\varepsilon_r = 1.5$ 倍を求める方法を説明する。 $\|[(e_m^B)^s b_0] - (e_m^B)^s b_0\|_2$ for $s = 1, 2$, and give the general result subsequently.

For $s = 1$, we have

$$\|[(e_m^B) b_0] - e_m^B b_0\|_2 < \stackrel{\text{(C17)}}{\beta} \|b_0\|_2 = \beta. \quad (\text{C19})$$

For $s = 2$, we obtain

$$\begin{aligned} & \|[(e_m^B)^2 b_0] - (e_m^B)^2 b_0\|_2 \\ & \leq \|[(e_m^B)^2 b_0] - e_m^B [(e_m^B) b_0]\|_2 + \|e_m^B [(e_m^B) b_0] - (e_m^B)^2 b_0\|_2 \\ & < \|[(e_m^B) [(e_m^B) b_0]] - e_m^B [(e_m^B) b_0]\|_2 + \beta \|e_m^B\|_2, \end{aligned} \quad (\text{C20})$$

ここで、最後のステップは式(C19)で可能になる。最後に、第2項の行列ノルムを次のように束縛する。

$$\begin{aligned} \|e_m^B\|_2 &= \|e^B - R_m(B)\|_2 \leq 1 + R_m(\|B\|_2) \\ &\stackrel{\text{(C4)}}{\leq} 1 + R_m(\|B\|_1) =: \alpha. \end{aligned} \quad (\text{C21})$$

式(C17)は任意の複素ベクトル b_0 に対して成立するので、式(C20)の誤差はさらに次のように制限される：

$$\begin{aligned} & \|[(e_m^B)^2 b_0] - (e_m^B)^2 b_0\|_2 < \stackrel{\text{(C17)}}{\beta} \|[(e_m^B) b_0]\|_2 + \beta \alpha \\ & = \beta \|[(e_m^B) b_0] - e_m^B b_0 + e_m^B b_0\|_2 + \beta \alpha \\ & < \beta(\beta + \alpha) + \beta \alpha. \end{aligned} \quad (\text{C22})$$

この戦略をさらに一般化すると、 ε_r の上限の式が得られる：

$$\begin{aligned} \varepsilon_r &= \|[(e_m^B)^s b_0] - (e_m^B)^s b_0\|_2 \\ &\leq \sum_{l=0}^{s-1} \|(e_m^B)^l [(e_m^B)^{s-l} b_0] - (e_m^B)^{l+1} [(e_m^B)^{s-l-1} b_0]\|_2 \\ &< \beta \sum_{l=0}^{s-1} (\alpha + \beta)^{s-l-1} \alpha^l = (\alpha + \beta)^s - \alpha^s. \end{aligned} \quad (\text{C23})$$

ε_r と ε_t の上界を組み合わせると、式(C1)に戻り、次のようになります。

$$\begin{aligned} \varepsilon_{A,|\Psi\rangle} &< E_A(m, s) = s R_m(\|B\|_1) \frac{1 - (s R_m(\|B\|_1))^s}{1 - s R_m(\|B\|_1)} \\ &\quad + (\alpha + \beta)^s - \alpha^s, \end{aligned} \quad (\text{C24})$$

where $\|B\|_1 = \|A\|_1 / s$, $\alpha = \alpha(\|A\|_1, m, s)$ and $\beta = \beta(\sigma', \|A\|_1, m, s)$.

Appendix D: Numerical comparison between scaling and squaring implementations

伝播状態積 $e^{-iH\Delta t} |\Psi\rangle$ を評価するためにスケーリングと二乗を使用する。我々のスケーリングと二乗の実装と Scipy の実装である expm_multiply [41]との相対誤差とランタイムを比較する。我々の実装と Scipy の実装の相対誤差は式(14)の一般式に従うが、 m と s の具体的な選択で異なる。正確な伝播状態積の代理として、Sympyによって400桁精度の行列指数にテイラー展開を適用すると、積の結果は小数点以下30桁で収束する。実行時間は Temci パッケージ [44] を用いて測定しています。この比較の基礎として、ハミルトニアン(20)と(21)の例を用い、それらにスパース記憶法を実装した。図5(a)と(c)に、それぞれ行列ノルム $\| -iH\Delta t \|_2$ に対する関数として、相対誤差を示す。どちらの例でも、我々の実装の相対誤差は誤差許容値 τ に拘束されるが、Scipy の実装の相対誤差は大きなノルムでは許容値を超える傾向にある。この傾向には2つの理由がある。第一に、丸め誤差は $\uparrow \uparrow \uparrow \uparrow$ の導出では考慮されていない。

per bound for the errors [41]; second the truncation of e_m^B [Eq. (12)] is merely based on a heuristic criterion.

図5(b)と(d)に示すように、我々の実装と Scipy の実装の μ の比は常に 1 より大きい。Scipy の実装は行列-ベクトル乗算に費やす時間は少ないが、図5(b)と(d)参照、総ランタイムは我々の実装より大きくなる。これは、Scipy の実装が (m, s) を決定するための実行時間オーバーヘッドが大きいためである。結論として、我々の実装は Scipy の実装よりも数値的な安定性と実行時間が優れていることが数値実験からわかった。

Appendix E: The scaling of μ

本文で議論したランタイムの漸近挙動は、ヒルベルト空間の次元 d に対する μ (16) のスケーリングによって支配されている。この依存性 $\mu(d)$ は指數化される行列 A (これはハミルトニアンに比例する) に特有のものである。このことは、最小化制約(15)の複雑さと相まって、一般的なスケーリング関係を述べることを妨げています。そこで、いくつかの具体的な例について議論することにする。まず、式(20)のようなトランスマンキャビティ系の例で、トランスマンのカットオフを固定したまま、キャビティの次元 d_c を経由して d を増加させます。対応するハミルトニアン H_1 は裸積ベースで書かれ、各行の非ゼロ要素の最大数が一定である。(すなわち、パラメータ σ^0 (C16) は d に依存しない)。数値的には、この場合、 μ は $|A|_1 \mu \propto \Theta(|A|_1)$ と線形にスケールし、この線形性は広い範囲の τ に対して成り立つことがわかります (図6(a) 参照)。

c. Upper bound for the error ε_r

The vector $(e_m^B)^s b_0$ is evaluated by the recursive relation.

$$c_l = e_m^B c_{l-1}, \quad l = 1, 2, \dots, s, \quad (\text{C18})$$

where $c_0 = b_0$. We illustrate how to bound the error $\varepsilon_r = \|\llbracket (e_m^B)^s b_0 \rrbracket - (e_m^B)^s b_0\|_2$ for $s = 1, 2$, and give the general result subsequently.

For $s = 1$, we have

$$\|\llbracket e_m^B b_0 \rrbracket - e_m^B b_0\|_2 \stackrel{(\text{C17})}{<} \beta \|b_0\|_2 = \beta. \quad (\text{C19})$$

For $s = 2$, we obtain

$$\begin{aligned} & \|\llbracket (e_m^B)^2 b_0 \rrbracket - (e_m^B)^2 b_0\|_2 \\ & \leq \|\llbracket (e_m^B)^2 b_0 \rrbracket - e_m^B \llbracket e_m^B b_0 \rrbracket\|_2 + \|e_m^B \llbracket e_m^B b_0 \rrbracket - (e_m^B)^2 b_0\|_2 \\ & < \|\llbracket e_m^B \llbracket e_m^B b_0 \rrbracket \rrbracket - e_m^B \llbracket e_m^B b_0 \rrbracket\|_2 + \beta \|e_m^B\|_2, \end{aligned} \quad (\text{C20})$$

where the last step is enabled by Eq. (C19). Finally, we bound the matrix norm in the second term by

$$\begin{aligned} \|e_m^B\|_2 &= \|e^B - R_m(B)\|_2 \leq 1 + R_m(\|B\|_2) \\ &\stackrel{(\text{C4})}{\leq} 1 + R_m(\|B\|_1) =: \alpha. \end{aligned} \quad (\text{C21})$$

Since Eq. (C17) holds for any complex vector b_0 , the error in Eq. (C20) is further bounded by:

$$\begin{aligned} & \|\llbracket (e_m^B)^2 b_0 \rrbracket - (e_m^B)^2 b_0\|_2 \stackrel{(\text{C17})}{<} \beta \|\llbracket e_m^B b_0 \rrbracket\|_2 + \beta \alpha \\ & = \beta \|\llbracket e_m^B b_0 \rrbracket - e_m^B b_0 + e_m^B b_0\|_2 + \beta \alpha \\ & \stackrel{(\text{C19})}{<} \beta(\beta + \alpha) + \beta \alpha. \end{aligned} \quad (\text{C22})$$

Generalizing this strategy further, we obtain the expression for the upper bound of ε_r :

$$\begin{aligned} \varepsilon_r &= \|\llbracket (e_m^B)^s b_0 \rrbracket - (e_m^B)^s b_0\|_2 \\ &\leq \sum_{l=0}^{s-1} \|(e_m^B)^l \llbracket (e_m^B)^{s-l} b_0 \rrbracket - (e_m^B)^{l+1} \llbracket (e_m^B)^{s-l-1} b_0 \rrbracket\|_2 \\ &< \beta \sum_{l=0}^{s-1} (\alpha + \beta)^{s-l-1} \alpha^l = (\alpha + \beta)^s - \alpha^s. \end{aligned} \quad (\text{C23})$$

Combining the upper bounds for ε_r and ε_t , we return to Eq. (C1) and obtain

$$\begin{aligned} \varepsilon_{A, |\Psi\rangle} &< E_A(m, s) = s R_m(\|B\|_1) \frac{1 - (s R_m(\|B\|_1))^s}{1 - s R_m(\|B\|_1)} \\ &\quad + (\alpha + \beta)^s - \alpha^s, \end{aligned} \quad (\text{C24})$$

where $\|B\|_1 = \|A\|_1 / s$, $\alpha = \alpha(\|A\|_1, m, s)$ and $\beta = \beta(\sigma', \|A\|_1, m, s)$.

Appendix D: Numerical comparison between scaling and squaring implementations

Scaling and squaring is used to evaluate the propagator-state product, $e^{-iH\Delta t}|\Psi\rangle$. We compare relative error and runtime between our implementation of scaling and squaring and `Scipy`'s implementation, `expm_multiply` [41]. The relative errors for our implementation and `Scipy`'s implementation follow the general expression in Eq. (14), but differ in specific choice of m and s . As a proxy for the exact propagator-state product, we apply Taylor expansion to matrix exponential with 400 digits precision by `Sympy` and the result of the product converges at 30 digits after decimal point. The runtime is measured with the help of the `Temci` package [44].

As the basis for this comparison, we use the examples of Hamiltonians (20) and (21) and implement sparse storage for them. The relative errors are shown in Fig. 5(a) and (c) as a function versus the matrix norm $\|-iH\Delta t\|_2$ respectively. In both examples, the relative errors for our implementation are bounded by the error tolerance τ , while the relative errors for `Scipy`'s implementation tend to exceed the tolerance for large norms. There are two reasons for this tendency. First, rounding errors are not considered in the derivation of the upper bound for the errors [41]; second the truncation of e_m^B [Eq. (12)] is merely based on a heuristic criterion.

Even though the relative errors of `Scipy`'s implementation are worse, it gains smaller number of matrix-vector multiplications, which is denoted by μ . This is illustrated in Fig. 5(b) and (d), that the ratio of μ between our implementation and `Scipy`'s implementation is always larger than 1. Although `Scipy`'s implementation spends less time on matrix-vector multiplications, the total runtime of `Scipy`'s implementation is larger than the one of our implementation, see Fig. 5(b) and (d). This is due to the larger runtime overhead of `Scipy`'s implementation for determining (m, s) .

In conclusion, our implementation has the better numerical stability and runtime than `Scipy`'s implementation in our numerical experiments.

Appendix E: The scaling of μ

The asymptotic behavior of runtime, discussed in the main text, is governed by the scaling of μ (16) with dimension d of the Hilbert space. The dependence $\mu(d)$ is specific to the matrix A to be exponentiated (which is proportional to the Hamiltonian). This together with the complexity of the minimization constraint (15) prevents us from stating a general scaling relation. We thus turn to a discussion of several concrete examples.

We first consider the example of the transmon-cavity system, see Eq. (20), where we increase d via the cavity dimension d_c while leaving the transmon cutoff fixed. The corresponding Hamiltonian H_1 , written in the bare product basis, then has a constant maximum number of non-zero elements in each row. (I.e., the parameter σ' (C16) is independent of d .) Numerically, we find in this case that μ scales linearly with

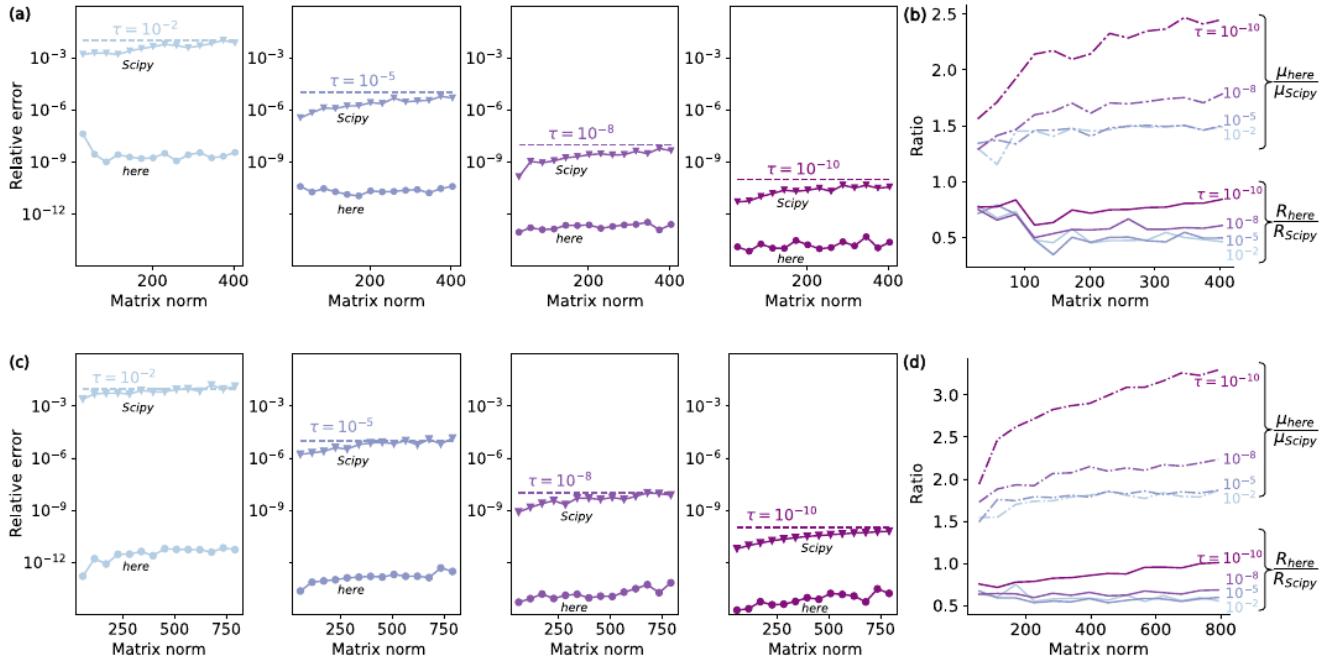


図5. 相対誤差と実行時間を考慮した、今回実装したスケーリングと2乗とScipyの`expm multiply`との比較。(a), (c) 相対誤差対行列ノルム。(b), (d) 行列ノルムの関数として、我々の実装とScipyの実装の間の実行時間と μ （すなわち、行列-ベクトル乗算の数）の比を示す。行列ノルム $\|A - iH\Delta t\alpha_2\|$ は α_2 を増加させることで変化させる [1, 10]。すべてのパネルで、色は異なる誤差許容値 τ を符号化している。1行目と2行目の結果は、それぞれキャビティトランスマントと3結合トランスマントのハミルトニアンに基づいて得られたものです。これらの2つのハミルトニアンのシステムパラメータは、それぞれ図2、図3のキャプションで与えられたものと同じです。

この例では、スケーリング $\|A\|_1 \leq \Theta(d^2)$ を導く2つの要素：(1) d は d_c の定数倍、(2) H_1 の空洞ラダー演算子は1ノルムスケーリング $\Theta(d^2)$ を持っています。したがって、図6(e)を参照すると、全体のスケーリング $\mu \leq \Theta(d^2)$ が得られることがわかります。第二の例では、式(21)を参照して、3つの結合したトランスマントの系を考える。ハミルトニアン H_2 は調和振動子基底で書かれ、定数 σ^0 を持つため、 $\mu \leq \Theta(\|A\|_1)$ となる [図6(b)]。 H_2 における演算子 $(b^\dagger b)^2$ の1ノルムは、各トランスマント次元で2次関数的にスケーリングされます。3つのトランスマントの次元を同時に大きくして d を増やすので、 $\|A\|_1 \leq \Theta(d^3)$ を維持することができる。このスケーリングの結果、 $\mu \leq \Theta(d^3)$ となる (図6(f) 参照)。 σ^0 が d に依存する他のハミルトニアン行列では、スケーリング $\mu \leq \Theta(\|A\|_1)$ がまだ成立する可能性がある。以下では、線形性が成立する例と、それが崩れる例の2つを具体的に示す。まず、最近接 σ_z 結合を持つ N_q 量子ビットの駆動系を考える。回転フレームにおいて、この系は次のように記述される。

$$H_3(t) = \sum_{\nu=1}^{N_q} [a_x^{(\nu)}(t)\sigma_x^{(\nu)} + a_y^{(\nu)}(t)\sigma_y^{(\nu)}] + \sum_{\nu=1}^{N_q-1} g\sigma_z^{(\nu)}\sigma_z^{(\nu+1)}.$$

(n) 各クビットは駆動場 a_x と a_y を介して制御される。この例では、数値計算の結果、スケーリング $\mu \leq \Theta(\|A\|_1)$ が依然として成り立つことがわかる (図6(c) 参照)。

N_q を大きくすることで d を大きくしているので、 $\|A\|_1 \leq \Theta(N_q) \leq \Theta(\log_2 d)$ が得られる。このスケーリングにより、 $\mu \leq \Theta(\log_2 d)$ が得られる (図6(g)を参照)。第二の例として、容量結合した二つのフラクソニウム量子ビットを考え、ハミルトニアン

$$H_4(t) = \sum_{i=1}^2 (4E_{Ci}n_i^2 - E_{Ji}\cos(\varphi_i) + \frac{E_{Li}}{2}[\varphi_i - \phi_i(t)])^2 + gn_1n_2. \quad (E1)$$

ここで、 E_{Ci} , E_{Ji} , E_{Li} は2つのフラクソニウム量子ビットの充電、誘導、ジョセフソンエネルギーを表し、 g は相互作用強度です。 ϕ と n は位相演算子と電荷数演算子で、通常の方法で定義する。ハミルトニアンを調和振動子基底で表現しています。図6(d)は、 $\tau = 10^{-8}$ のとき、 μ のスケーリングが $\|A\|_1$ と超線形であることを示す。そこで、フラクソニウム量子ビットの次元切断を同時に大きくして、 $\tau = 10^{-8}$ における μ と d のスケーリング関係を数値的に決定することにした。その結果、図6(h)に示すように、 $\mu \leq \Theta(d^{0.65})$ のスケーリングが観測された。

付録F: HGスキームの代替数値実装と対応するスケーリング解析

ハードコード勾配法の数値実装には、短時間伝搬子とその微分値の評価が必要である。ある条件下では、行列対角化を用いて短時間伝搬子を数値的に評価することが有利で

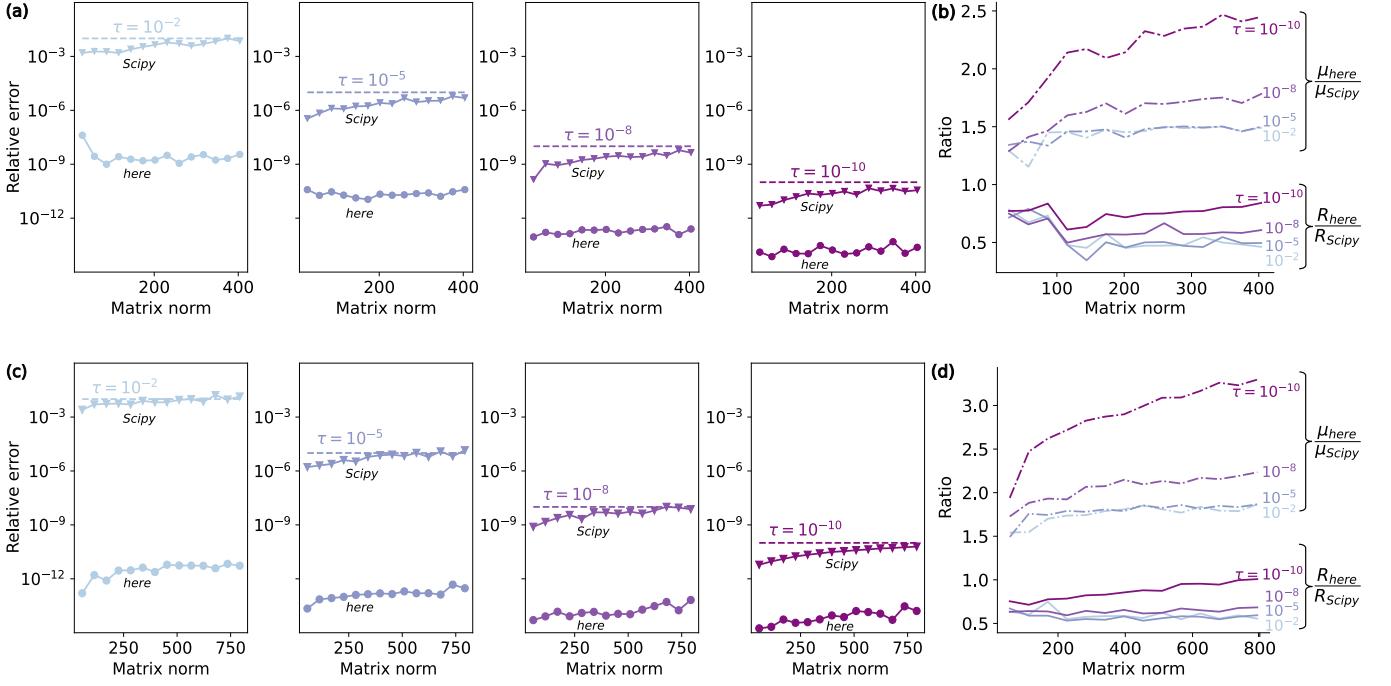


Figure 5. Comparison between scaling and squaring as implemented here and Scipy’s `expm_multiply`, considering relative error and runtime. (a), (c) Relative error versus matrix norm. (b), (d) Ratios of runtime and μ (i.e. number of matrix-vector multiplications) between our and Scipy’s implementations as functions of the matrix norm. The matrix norm $\| -iH\Delta t \|_2$ is varied by increasing $\Delta t \in [1, 10]$. In all panels, colors encode different error tolerances τ . The results of the first and second row are obtained based on cavity-transmon and three coupled transmons Hamiltonians, respectively. The system parameters of these two Hamiltonian are the same as the ones given in the captions of Figs. 2, 3, respectively.

$\|A\|_1$ ($\mu \sim \Theta(\|A\|_1)$), and this linearity holds for a wide range of τ , see Fig. 6(a). In this example, two ingredients lead to the scaling $\|A\|_1 \sim \Theta(d^{\frac{1}{2}})$: (1) d is a constant multiple of d_c , and (2) the cavity ladder operators in H_1 which has one-norm scaling $\Theta(d_c^{\frac{1}{2}})$. Thus, we obtain the overall scaling $\mu \sim \Theta(d^{\frac{1}{2}})$, see Fig. 6(e). In the second example, we consider the system three coupled transmons, see Eq. (21). The Hamiltonian H_2 is written in the harmonic oscillator basis and has a constant σ' , which leads to $\mu \sim \Theta(\|A\|_1)$ [Fig. 6(b)]. One-norm of operator $(b^\dagger b)^2$ in H_2 has quadratic scaling with each transmon dimension. Since we increase d by enlarging the dimensions of all three transmons simultaneously, we obtain $\|A\|_1 \sim \Theta(d^{\frac{2}{3}})$. This scaling results in $\mu \sim \Theta(d^{\frac{2}{3}})$, see Fig. 6(f).

For other Hamiltonian matrices where σ' depends on d , the scaling $\mu \sim \Theta(\|A\|_1)$ may still hold. In the following, we show two concrete examples that one where linearity holds, and another where it breaks. We first consider a driven system of N_q qubits with nearest-neighbor σ_z coupling. In the rotating frame, the system is described by

$$H_3(t) = \sum_{\nu=1}^{N_q} [a_x^{(\nu)}(t)\sigma_x^{(\nu)} + a_y^{(\nu)}(t)\sigma_y^{(\nu)}] + \sum_{\nu=1}^{N_q-1} g\sigma_z^{(\nu)}\sigma_z^{(\nu+1)}.$$

Each qubit is controlled via drive fields $a_x^{(n)}$ and $a_y^{(n)}$. For this example, the numerical results show that scaling $\mu \sim \Theta(\|A\|_1)$ still holds, see Fig. 6(c). Since we increase d by

enlarging N_q , we obtain $\|A\|_1 \sim \Theta(N_q) \sim \Theta(\log_2 d)$. This scaling results in $\mu \sim \Theta(\log_2 d)$, see Fig. 6(g). For the second example, we consider two capacitively coupled fluxonium qubits with Hamiltonian

$$H_4(t) = \sum_{i=1}^2 (4E_{Ci}n_i^2 - E_{Ji}\cos(\varphi_i) + \frac{E_{Li}}{2}[\varphi_i - \phi_i(t)]^2) + gn_1n_2. \quad (\text{E1})$$

Here, E_{Ci} , E_{Ji} , E_{Li} denote the charging, inductive and Josephson energies of two fluxonium qubits, and g is the interaction strength. φ and n are the phase and charge number operators, defined in the usual way. We represent the Hamiltonian in the harmonic oscillator basis. Fig. 6(d) shows that scaling of μ is superlinear with $\|A\|_1$ for $\tau = 10^{-8}$. We thus settle for a numerical determination of the scaling relation between μ and d for $\tau = 10^{-8}$ by increasing the dimensional cutoff for both fluxonium qubits simultaneously. The observed scaling is $\mu \sim \Theta(d^{0.65})$, see Fig. 6(h).

Appendix F: Alternative numerical implementation of HG scheme and the corresponding scaling analysis

The numerical implementation of the hard-coded gradient scheme requires evaluation of the short-time propagator and its derivative. Under certain conditions, it turns out to be advantageous to numerically evaluate the short-time propagator

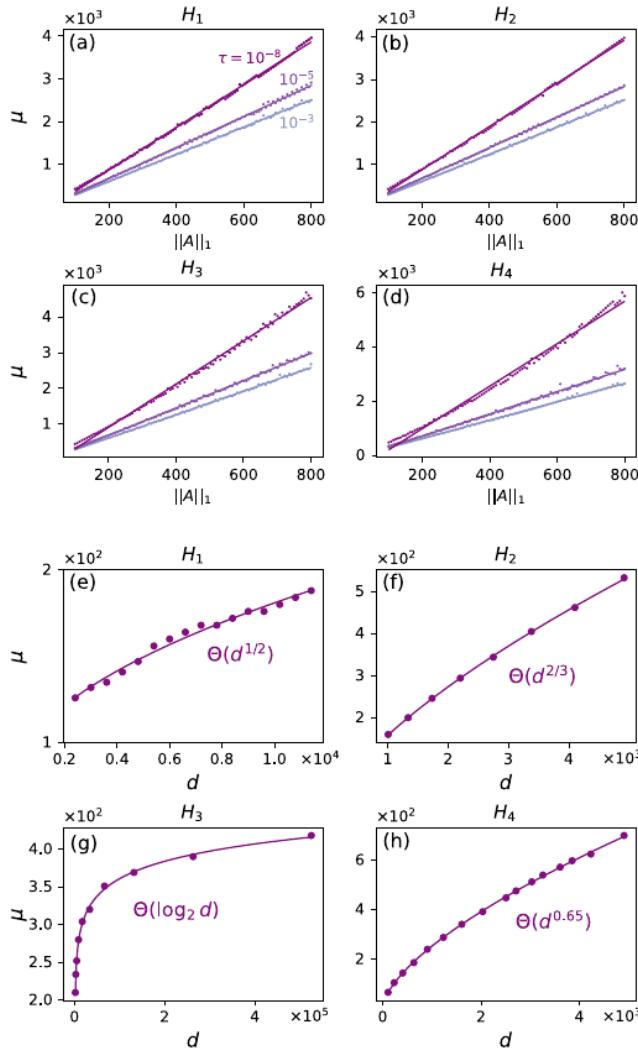


図6. 行列-ベクトル乗算の回数である μ のスケーリング。(a)-(d) ハミルトニアン H_j , $j = 1, 2, 3, 4$ に対して、いくつかの誤差許容度 τ を考慮した場合の μ_{Axm1} による μ のスケーリング。(e)-(h) 各 H_j に対してそれぞれ $\tau = 10^{-8}$ と考えた場合の次元 d に対する μ のスケーリング。図中、ドットはサンプリングデータ、実線はフィッティングカーブである。 H_1 , H_2 のパラメータ設定は、図 2, 3 のキャプションにあるものと同じである。 H_3 , H_4 のパラメータは以下の通り： H_3 の場合、 $E_{Cj} / 2\pi = 2.5 \text{ GHz}$, $E_{jj} / 2\pi = 8.9 \text{ GHz}$, $E_{Lj} / 2\pi = 0.5 \text{ GHz}$, $g / 2\pi = 0.1 \text{ GHz}$, $\phi = 0.33$ ； H_4 の場合、 $a_x^{\text{mix}} / 2\pi = a_y^{\text{mix}} / 2\pi = 0.5 \text{ GHz}$, $g / 2\pi = 0.1 \text{ GHz}$ です。

あることが判明している。本付録では、まず伝搬体微分の計算方法について説明し、次にHGのスケーリングについて解析する。

1. Evaluation of propagator derivative

演算子 $-iH$ は反エルミートであるため、次のように対角化し指数化することができる。

$$A = SDS^\dagger, \quad (\text{F1})$$

$$e^A = Se^D S^\dagger. \quad (\text{F2})$$

ここで、 S はユニタリー、 D はダイアゴナルである。その結果、伝搬体微分は次のように書くことができる。

$$\frac{de^A}{da} = Se^D \frac{dD}{da} S^\dagger + \frac{dS}{da} e^D S^\dagger + Se^D \frac{dS^\dagger}{da}. \quad (\text{F3})$$

We apply the inverse unitary transformation to $\frac{de^A}{da}$ (F3) and obtain

$$S^\dagger \frac{de^A}{da} S = e^D \frac{dD}{da} + S^\dagger \frac{dS}{da} e^D + e^D \frac{dS^\dagger}{da} S. \quad (\text{F4})$$

Based on $\frac{dS^\dagger}{da} S + S^\dagger \frac{dS}{da} = 0$, Eq. (F4) can be rewritten as

$$S^\dagger \frac{de^A}{da} S = e^D \frac{dD}{da} + E \circ (S^\dagger \frac{dS}{da}) \quad (\text{F5})$$

とし、 $E_{ij} := e^D j j - e^D i i$ 。ここで、 \circ はハダマード積(要素ごとの乗算)を表す。 dD/da , dS/da を求めるには、式 (F1) を微分し、式 (F4)、(F5) と同じ手順を繰り返すと、次のようにになります。

$$S^\dagger \frac{dA}{da} S = \frac{dD}{da} + E' \circ (S^\dagger \frac{dS}{da}), \quad (\text{F6})$$

ここで、 $E_{ij}^0 := D_{jj} - D_{ii}$ 。 E^0 の対角要素は0であるから、次のようになる。

$$\frac{dD}{da} = I \circ (S^\dagger \frac{dA}{da} S). \quad (\text{F7})$$

式(F6)の演算子の非対角要素について、次のようになります。

$$F \circ (S^\dagger \frac{dA}{da} S) + G = S^\dagger \frac{dS}{da}, \quad (\text{F8})$$

where we define

$$F_{ij} := \begin{cases} \frac{1}{D_{jj} - D_{ii}}, & \text{if } D_{jj} - D_{ii} \neq 0. \\ 0, & \text{otherwise.} \end{cases} \quad (\text{F9})$$

$$G_{ij} := \begin{cases} 0, & \text{if } D_{jj} - D_{ii} \neq 0. \\ (S^\dagger \frac{dS}{da})_{ij}, & \text{otherwise.} \end{cases}$$

Inserting Eqs. (F7) and (F8) into Eq. (F5) gives

$$\frac{de^A}{da} = S \left((e^D + E \circ F) \circ (S^\dagger \frac{dA}{da} S) \right) S^\dagger, \quad (\text{F10})$$

where we use $E \circ G = 0$.

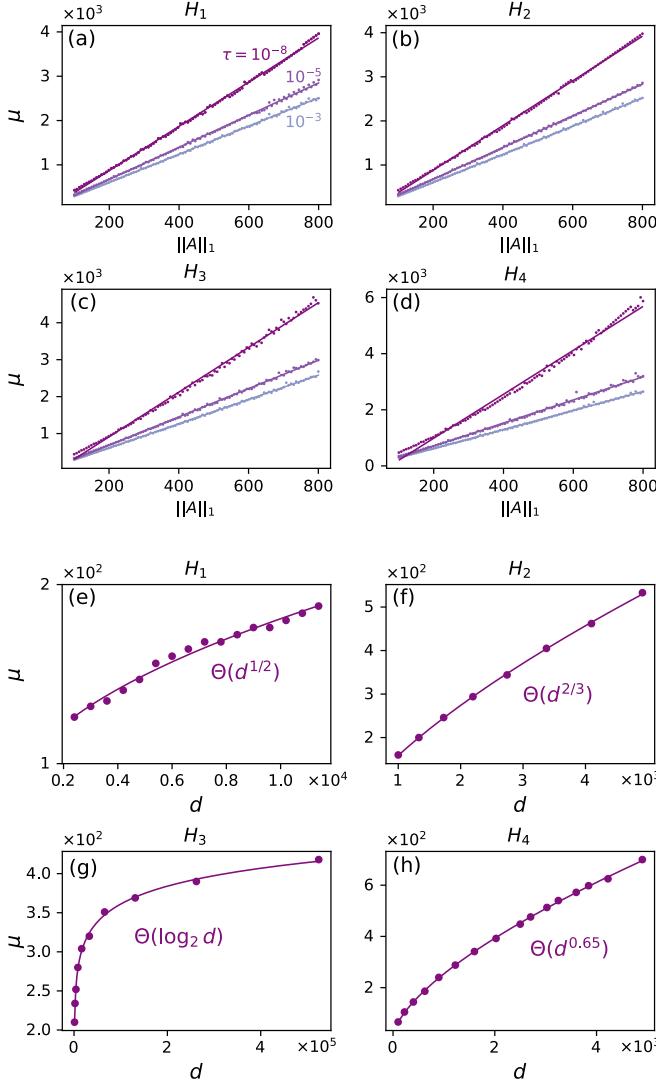


Figure 6. Scaling of μ , the number of matrix-vector multiplications. (a)-(d) The scaling of μ with $\|A\|_1$ for Hamiltonians H_j , $j = 1, 2, 3, 4$, when considering several error tolerances τ . (e)-(h) The scaling of μ with dimension d when considering $\tau = 10^{-8}$ for each H_j , respectively. In all figures, dots are sampling data and solid lines are fitting curves. The parameters setup of H_1 , H_2 are the same as the ones given in the captions of Figs. 2, 3. Parameters of H_3 and H_4 are as follows: for H_3 , $E_{Ci}/2\pi = 2.5$ GHz, $E_{Ji}/2\pi = 8.9$ GHz, $E_{Li}/2\pi = 0.5$ GHz, $g/2\pi = 0.1$ GHz and $\phi = 0.33$; for H_4 , $a_x^{(\nu)}/2\pi = a_y^{(\nu)}/2\pi = 0.5$ GHz and $g/2\pi = 0.1$ GHz.

by employing matrix diagonalization. In this appendix, we first discuss the method to compute propagator derivative and then analyze the scaling of HG.

1. Evaluation of propagator derivative

The operator $-iH\Delta t$ is anti-Hermitian, and hence can be diagonalized and exponentiated according to

$$A = SDS^\dagger, \quad (\text{F1})$$

$$e^A = Se^D S^\dagger. \quad (\text{F2})$$

Here, S is unitary and D diagonal. As a result, the propagator derivative can be written as

$$\frac{de^A}{da} = Se^D \frac{dD}{da} S^\dagger + \frac{dS}{da} e^D S^\dagger + Se^D \frac{dS^\dagger}{da}. \quad (\text{F3})$$

We apply the inverse unitary transformation to $\frac{de^A}{da}$ (F3) and obtain

$$S^\dagger \frac{de^A}{da} S = e^D \frac{dD}{da} + S^\dagger \frac{dS}{da} e^D + e^D \frac{dS^\dagger}{da} S. \quad (\text{F4})$$

Based on $\frac{dS^\dagger}{da} S + S^\dagger \frac{dS}{da} = 0$, Eq. (F4) can be rewritten as

$$S^\dagger \frac{de^A}{da} S = e^D \frac{dD}{da} + E \circ (S^\dagger \frac{dS}{da}) \quad (\text{F5})$$

with $E_{ij} := e^{D_{jj}} - e^{D_{ii}}$. Here, \circ denotes the Hadamard product (element-wise multiplication). To obtain dD/da and dS/da , we take the derivative of Eq. (F1) and repeat the same steps as for Eqs. (F4) and (F5), leading to

$$S^\dagger \frac{dA}{da} S = \frac{dD}{da} + E' \circ (S^\dagger \frac{dS}{da}), \quad (\text{F6})$$

where $E'_{ij} := D_{jj} - D_{ii}$. Since the diagonal elements of E' are zero, it follows that

$$\frac{dD}{da} = I \circ (S^\dagger \frac{dA}{da} S). \quad (\text{F7})$$

For off-diagonal elements of the operator in Eq. (F6), we obtain

$$F \circ (S^\dagger \frac{dA}{da} S) + G = S^\dagger \frac{dS}{da}, \quad (\text{F8})$$

where we define

$$F_{ij} := \begin{cases} \frac{1}{D_{jj} - D_{ii}}, & \text{if } D_{jj} - D_{ii} \neq 0, \\ 0, & \text{otherwise.} \end{cases} \quad (\text{F9})$$

$$G_{ij} := \begin{cases} 0, & \text{if } D_{jj} - D_{ii} \neq 0, \\ (S^\dagger \frac{dS}{da})_{ij}, & \text{otherwise.} \end{cases}$$

Inserting Eqs. (F7) and (F8) into Eq. (F5) gives

$$\frac{de^A}{da} = S((e^D + E \circ F) \circ (S^\dagger \frac{dA}{da} S)) S^\dagger, \quad (\text{F10})$$

where we use $E \circ G = 0$.

2. Scaling analysis

コスト関数への寄与の勾配（状態遷移の場合）は、前方後方伝搬方式で評価される。この方式では、状態に必要なストレージは中間時間ステップの数に対してスケールしません。保存しなければならない行列は、システムと制御のハミルトニアン、および密な行列dense matrices S 、 E 、 E^0 である。したがって、総メモリ使用量は $O(d^2)$ の規模になる。

このスキームの実行時間は、伝搬体とその微分の $O(N)$ 個の評価によって支配される。伝搬体の評価に行列の対角化を用いると、実行時間は $O(d^3)$ になる[53]。伝搬体微分の計算には、行列と行列の乗算とハダマルド積が必要で、それぞれ $O(d^3)$ 、 $O(d^2)$ の実行時間スケーリングが必要です。従って、HGスキーム全体の実行時間は $O(N d^3)$ のスケールとなる。ゲート操作コスト関数への寄与の勾配も同様の方法で評価され、実行時間とメモリのスケーリングが実現されます。

-
- [1] K. Rojan, D. M. Reich, I. Dotsenko, J.-M. Raimond, C. P. Koch, and G. Morigi, *Physical Review A* **90**, 023824 (2014).
 - [2] G. Waldherr, Y. Wang, S. Zaiser, *et al.*, *Nature* **506**, 204 (2014).
 - [3] J. M. Gertler, B. Baker, J. Li, S. Shirol, J. Koch, and C. Wang, *Nature* **590**, 243 (2021).
 - [4] M. Abdelhafez, B. Baker, A. Gyenis, P. Mundada, A. A. Houck, D. Schuster, and J. Koch, *Physical Review A* **101**, 022321 (2020).
 - [5] J. L. Allen, R. Kosut, J. Joo, P. Leek, and E. Ginossar, *Physical Review A* **95**, 042325 (2017).
 - [6] S.-Y. Huang and H.-S. Goan, *Physical Review A* **90**, 012318 (2014).
 - [7] M. Werninghaus, D. J. Egger, F. Roy, S. Machnes, F. K. Wilhelm, and S. Filipp, *npj Quantum Information* **7**, 14 (2021).
 - [8] S. J. Glaser, U. Boscain, *et al.*, *The European Physical Journal D* **69**, 279 (2015).
 - [9] C. Bretschneider, A. Karabanov, N. C. Nielsen, and W. Köckenberger, *The Journal of Chemical Physics* **136**, 094201 (2012).
 - [10] C. T. Kehlet, A. C. Sivertsen, M. Bjerring, T. O. Reiss, N. Khaneja, S. J. Glaser, and N. C. Nielsen, *Journal of the American Chemical Society* **126**, 10202 (2004).
 - [11] N. Khaneja, T. Reiss, C. Kehlet, T. Schulte-Herbrüggen, and S. J. Glaser, *Journal of Magnetic Resonance* **172**, 296 (2005).
 - [12] Z. Tošner, T. Vosegaard, C. Kehlet, N. Khaneja, S. J. Glaser, and N. C. Nielsen, *Journal of Magnetic Resonance* **197**, 120 (2009).
 - [13] H. Hogben, M. Krzystyniak, G. Charnock, P. Hore, and I. Kuprov, *Journal of Magnetic Resonance* **208**, 179 (2011).
 - [14] D. Xu, K. F. King, Y. Zhu, G. C. McKinnon, and Z.-P. Liang, *Magnetic Resonance in Medicine* **59**, 547 (2008).
 - [15] M. M. Müller, U. G. Poschinger, T. Calarco, S. Montangero, and F. Schmidt-Kaler, *Physical Review A* **92**, 053423 (2015).
 - [16] V. Nebendahl, H. Häffner, and C. F. Roos, *Physical Review A* **79**, 012312 (2009).
 - [17] A. Angerer, *Robust coherent optimal control of nitrogen-vacancy quantum bits in diamond*, Ph.D. thesis, Vienna University of Technology (2013).
 - [18] Y. Chou, S.-Y. Huang, and H.-S. Goan, *Physical Review A* **91**, 052315 (2015).
 - [19] F. Dolde, V. Bergholm, Y. Wang, I. Jakobi, B. Naydenov, S. Pezzagna, J. Meijer, F. Jelezko, P. Neumann, T. Schulte-Herbrüggen, J. Biamonte, and J. Wrachtrup, *Nature Communications* **5**, 3371 (2014).
 - [20] F. Poggiali, P. Cappellaro, and N. Fabbri, *Physical Review X* **8**, 021059 (2018).
 - [21] A. F. L. Poulsen, J. D. Clement, J. L. Webb, R. H. Jensen, L. Troise, K. Berg-Sørensen, A. Huck, and U. L. Andersen, *Physical Review B* **106**, 014202 (2022).
 - [22] P. Rembold, N. Oshnik, M. M. Müller, S. Montangero, T. Calarco, and E. Neu, *AVS Quantum Science* **2**, 024701 (2020).
 - [23] R.-Y. Yan and Z.-B. Feng, *JETP Letters* **114**, 314 (2021).
 - [24] S. Amri, R. Corgier, D. Sugny, E. M. Rasel, N. Gaaloul, and E. Charron, *Scientific Reports* **9**, 5346 (2019).
 - [25] S. Amri, R. Corgier, E. Rasel Maria, E. Charron, N. Gaaloul, and Q. Team, in *APS Division of Atomic, Molecular and Optical Physics Meeting Abstracts*, Vol. 65 (2020).
 - [26] J.-F. Mennemann, D. Matthes, R.-M. Weishäupl, and T. Langen, *New Journal of Physics* **17**, 113027 (2015).
 - [27] J. Sørensen, J. Jensen, T. Heinzel, and J. Sherson, *Computer Physics Communications* **243**, 135 (2019).
 - [28] J. Guo, X. Feng, P. Yang, Z. Yu, L. Chen, C.-H. Yuan, and W. Zhang, *Nature Communications* **10**, 148 (2019).
 - [29] S. Rosi, A. Bernard, N. Fabbri, L. Fallani, C. Fort, M. Inguscio, T. Calarco, and S. Montangero, *Physical Review A* **88**, 021601 (2013).
 - [30] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *Journal of Machine Learning Research* **18**, 1 (2018).
 - [31] N. Leung, M. Abdelhafez, J. Koch, and D. Schuster, *Physical Review A* **95**, 042318 (2017).
 - [32] T. Propson, “Qoc,” <https://github.com/SchusterLab/qoc.git> (2019).
 - [33] M. Abdelhafez, D. I. Schuster, and J. Koch, *Physical Review A* **99**, 052327 (2019).
 - [34] M. H. Goerz, S. C. Carrasco, and V. S. Malinovsky, *Quantum* **6**, 871 (2022).
 - [35] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, *et al.*, *Nature* **574**, 505 (2019).
 - [36] M. Gong, S. Wang, C. Zha, M.-C. Chen, H.-L. Huang, Y. Wu, Q. Zhu, Y. Zhao, S. Li, S. Guo, *et al.*, *Science* **372**, 948 (2021).
 - [37] M. Abadi *et al.*, “TensorFlow: Large-scale machine learning on heterogeneous systems,” (2015), software available from tensorflow.org.
 - [38] D. Maclaurin, D. Duvenaud, and R. P. Adams, in *ICML 2015 AutoML workshop*, Vol. 238 (2015).
 - [39] C. Moler and C. Van Loan, *SIAM Review* **45**, 3 (2003).
 - [40] B. Codenotti and C. Fassino, *Calcolo* **29**, 1 (1992).
 - [41] A. H. Al-Mohy and N. J. Higham, *SIAM Journal on Scientific Computing* **33**, 488 (2011).
 - [42] D. Goodwin and I. Kuprov, *The Journal of Chemical Physics* **143**, 084113 (2015).
 - [43] I. Najfeld and T. Havel, *Advances in Applied Mathematics* **16**, 321 (1995).
 - [44] J. Bechberger, “Temci,” <https://github.com/parttimenerd/temci.git> (2015).

2. Scaling analysis

The gradients of the contributions to the cost function (for the case of state transfer) are evaluated by the forward-backward propagation scheme. In this scheme, the storage required for states does not scale with the number of intermediate time steps. The matrices that must be stored are the system and control Hamiltonians as well as dense matrices dense matrices S , E and E' . Thus, the total memory usage scales as $O(d^2)$. The runtime of the scheme is dominated by $O(N)$

evaluations of the propagator and its derivative. Employing matrix diagonalization for the propagator evaluation requires a runtime scaling $\sim O(d^3)$ [53]. Calculation of the propagator derivative involves matrix-matrix multiplication and Hadamard product with runtime scaling $O(d^3)$ and $O(d^2)$, respectively. Thus, the overall runtime of the HG scheme scales as $O(Nd^3)$.

The gradients of the contributions to the gate-operation cost function are evaluated in an analogous manner, resulting in the runtime and memory scaling.

-
- [1] K. Rojan, D. M. Reich, I. Dotsenko, J.-M. Raimond, C. P. Koch, and G. Morigi, *Physical Review A* **90**, 023824 (2014).
 - [2] G. Waldherr, Y. Wang, S. Zaiser, *et al.*, *Nature* **506**, 204 (2014).
 - [3] J. M. Gertler, B. Baker, J. Li, S. Shirol, J. Koch, and C. Wang, *Nature* **590**, 243 (2021).
 - [4] M. Abdelhafez, B. Baker, A. Gyenis, P. Mundada, A. A. Houck, D. Schuster, and J. Koch, *Physical Review A* **101**, 022321 (2020).
 - [5] J. L. Allen, R. Kosut, J. Joo, P. Leek, and E. Ginossar, *Physical Review A* **95**, 042325 (2017).
 - [6] S.-Y. Huang and H.-S. Goan, *Physical Review A* **90**, 012318 (2014).
 - [7] M. Werninghaus, D. J. Egger, F. Roy, S. Machnes, F. K. Wilhelm, and S. Filipp, *npj Quantum Information* **7**, 14 (2021).
 - [8] S. J. Glaser, U. Boscain, *et al.*, *The European Physical Journal D* **69**, 279 (2015).
 - [9] C. Bretschneider, A. Karabanov, N. C. Nielsen, and W. Köckenberger, *The Journal of Chemical Physics* **136**, 094201 (2012).
 - [10] C. T. Kehlet, A. C. Sivertsen, M. Bjerring, T. O. Reiss, N. Khaneja, S. J. Glaser, and N. C. Nielsen, *Journal of the American Chemical Society* **126**, 10202 (2004).
 - [11] N. Khaneja, T. Reiss, C. Kehlet, T. Schulte-Herbrüggen, and S. J. Glaser, *Journal of Magnetic Resonance* **172**, 296 (2005).
 - [12] Z. Tošner, T. Vosegaard, C. Kehlet, N. Khaneja, S. J. Glaser, and N. C. Nielsen, *Journal of Magnetic Resonance* **197**, 120 (2009).
 - [13] H. Hogben, M. Krzystyniak, G. Charnock, P. Hore, and I. Kuprov, *Journal of Magnetic Resonance* **208**, 179 (2011).
 - [14] D. Xu, K. F. King, Y. Zhu, G. C. McKinnon, and Z.-P. Liang, *Magnetic Resonance in Medicine* **59**, 547 (2008).
 - [15] M. M. Müller, U. G. Poschinger, T. Calarco, S. Montangero, and F. Schmidt-Kaler, *Physical Review A* **92**, 053423 (2015).
 - [16] V. Nebendahl, H. Häffner, and C. F. Roos, *Physical Review A* **79**, 012312 (2009).
 - [17] A. Angerer, *Robust coherent optimal control of nitrogen-vacancy quantum bits in diamond*, Ph.D. thesis, Vienna University of Technology (2013).
 - [18] Y. Chou, S.-Y. Huang, and H.-S. Goan, *Physical Review A* **91**, 052315 (2015).
 - [19] F. Dolde, V. Bergholm, Y. Wang, I. Jakobi, B. Naydenov, S. Pezzagna, J. Meijer, F. Jelezko, P. Neumann, T. Schulte-Herbrüggen, J. Biamonte, and J. Wrachtrup, *Nature Communications* **5**, 3371 (2014).
 - [20] F. Poggiali, P. Cappellaro, and N. Fabbri, *Physical Review X* **8**, 021059 (2018).
 - [21] A. F. L. Poulsen, J. D. Clement, J. L. Webb, R. H. Jensen, L. Troise, K. Berg-Sørensen, A. Huck, and U. L. Andersen, *Physical Review B* **106**, 014202 (2022).
 - [22] P. Rembold, N. Oshnik, M. M. Müller, S. Montangero, T. Calarco, and E. Neu, *AVS Quantum Science* **2**, 024701 (2020).
 - [23] R.-Y. Yan and Z.-B. Feng, *JETP Letters* **114**, 314 (2021).
 - [24] S. Amri, R. Corgier, D. Sugny, E. M. Rasel, N. Gaaloul, and E. Charron, *Scientific Reports* **9**, 5346 (2019).
 - [25] S. Amri, R. Corgier, E. Rasel Maria, E. Charron, N. Gaaloul, and Q. Team, in *APS Division of Atomic, Molecular and Optical Physics Meeting Abstracts*, Vol. 65 (2020).
 - [26] J.-F. Mennemann, D. Matthes, R.-M. Weishäupl, and T. Langen, *New Journal of Physics* **17**, 113027 (2015).
 - [27] J. Sørensen, J. Jensen, T. Heinzel, and J. Sherson, *Computer Physics Communications* **243**, 135 (2019).
 - [28] J. Guo, X. Feng, P. Yang, Z. Yu, L. Chen, C.-H. Yuan, and W. Zhang, *Nature Communications* **10**, 148 (2019).
 - [29] S. Rosi, A. Bernard, N. Fabbri, L. Fallani, C. Fort, M. Inguscio, T. Calarco, and S. Montangero, *Physical Review A* **88**, 021601 (2013).
 - [30] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *Journal of Machine Learning Research* **18**, 1 (2018).
 - [31] N. Leung, M. Abdelhafez, J. Koch, and D. Schuster, *Physical Review A* **95**, 042318 (2017).
 - [32] T. Propson, “Qoc,” <https://github.com/SchusterLab/qoc.git> (2019).
 - [33] M. Abdelhafez, D. I. Schuster, and J. Koch, *Physical Review A* **99**, 052327 (2019).
 - [34] M. H. Goerz, S. C. Carrasco, and V. S. Malinovsky, *Quantum* **6**, 871 (2022).
 - [35] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, *et al.*, *Nature* **574**, 505 (2019).
 - [36] M. Gong, S. Wang, C. Zha, M.-C. Chen, H.-L. Huang, Y. Wu, Q. Zhu, Y. Zhao, S. Li, S. Guo, *et al.*, *Science* **372**, 948 (2021).
 - [37] M. Abadi *et al.*, “TensorFlow: Large-scale machine learning on heterogeneous systems,” (2015), software available from tensorflow.org.
 - [38] D. Maclaurin, D. Duvenaud, and R. P. Adams, in *ICML 2015 AutoML workshop*, Vol. 238 (2015).
 - [39] C. Moler and C. Van Loan, *SIAM Review* **45**, 3 (2003).
 - [40] B. Codenotti and C. Fassino, *Calcolo* **29**, 1 (1992).
 - [41] A. H. Al-Mohy and N. J. Higham, *SIAM Journal on Scientific Computing* **33**, 488 (2011).
 - [42] D. Goodwin and I. Kuprov, *The Journal of Chemical Physics* **143**, 084113 (2015).
 - [43] I. Najfeld and T. Havel, *Advances in Applied Mathematics* **16**, 321 (1995).
 - [44] J. Bechberger, “Temci,” <https://github.com/parttimenerd/temci.git> (2015).

- [45] F. Pedregosa, “Memory profiler,” https://github.com/pythonprofilers/memory_profiler.git (2012).
- [46] J. D. Hunter, *Computing in Science & Engineering* **9**, 90 (2007).
- [47] C. R. Harris, K. J. Millman, S. J. Van Der Walt, *et al.*, *Nature* **585**, 357 (2020).
- [48] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, *Nature Methods* **17**, 261 (2020).
- [49] A. Meurer, C. P. Smith, M. Paprocki, *et al.*, *PeerJ Computer Science* **3**, e103 (2017).
- [50] P. Groszkowski and J. Koch, *Quantum* **5**, 583 (2021).
- [51] N. J. Higham, *Accuracy and stability of numerical algorithms*, 4th ed. (SIAM, 2002) Chap. 2–3.
- [52] J. H. Wilkinson, *Rounding errors in algebraic processes*, (Courier Corporation, 1994) p. 82.
- [53] V. Y. Pan and Z. Q. Chen, in *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, STOC '99 (Association for Computing Machinery, 1999) p. 507–516.

- [45] F. Pedregosa, “Memory profiler,” https://github.com/pythonprofilers/memory_profiler.git (2012).
- [46] J. D. Hunter, *Computing in Science & Engineering* **9**, 90 (2007).
- [47] C. R. Harris, K. J. Millman, S. J. Van Der Walt, *et al.*, *Nature* **585**, 357 (2020).
- [48] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, *Nature Methods* **17**, 261 (2020).
- [49] A. Meurer, C. P. Smith, M. Paprocki, *et al.*, *PeerJ Computer Science* **3**, e103 (2017).
- [50] P. Groszkowski and J. Koch, *Quantum* **5**, 583 (2021).
- [51] N. J. Higham, *Accuracy and stability of numerical algorithms*, 4th ed. (SIAM, 2002) Chap. 2–3.
- [52] J. H. Wilkinson, *Rounding errors in algebraic processes*, (Courier Corporation, 1994) p. 82.
- [53] V. Y. Pan and Z. Q. Chen, in *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, STOC ’99 (Association for Computing Machinery, 1999) p. 507–516.