Quantum Optimal Control via Semi-Automatic Differentiation

Michael H. Goerz, Sebastián C. Carrasco, and Vladimir S. Malinovsky

DEVCOM Army Research Laboratory, 2800 Powder Mill Road, Adelphi, MD 20783, USA

We develop a framework of "semi-automatic differentiation" that combines existing gradient-based methods of quantum optimal control with automatic differentiation. The approach allows to optimize practically any computable functional and is implemented in two open source Julia packages, `GRAPE.jl` and `Krotov.jl`, part of the `QuantumControl.jl` framework. Our method is based on formally rewriting the optimization functional in terms of propagated states, overlaps with target states, or quantum gates. An analytical application of the chain rule then allows to separate the time propagation and the evaluation of the functional when calculating the gradient. The former can be evaluated with great efficiency via a modified GRAPE scheme. The latter is evaluated with automatic differentiation, but with a profoundly reduced complexity compared to the time propagation. Thus, our approach eliminates the prohibitive memory and runtime overhead normally associated with automatic differentiation and facilitates further advancement in quantum control by enabling the direct optimization of non-analytic functionals for quantum information and quantum metrology, especially in open quantum systems. We illustrate and benchmark the use of semi-automatic differentiation for the optimization of perfectly entangling quantum gates on superconducting qubits coupled via a shared transmission line. This includes the first direct optimization of the non-analytic gate concurrence.

1 Introduction

Optimal control is a cornerstone in the development of quantum technologies [1–7]. Quantum information processing [8], quantum simulation [9], and quantum sensing [10] all rely on the ability to manipulate matter at the fundamental quantum level. In simple cases, analytical solutions to the control problem may be possible [11]. In more realistic settings, in particular, when taking into account classical or quantum noise, numerical optimization methods must be used. The most general methods for open-loop pulse-level control, gradient-ascent-pulse-engineering (GRAPE) [12, 13] and Krotov's method [14–20] can both harness the full range of control possible via arbitrary waveform generators [21] or optical pulse shapers [22]. These methods iteratively update the controls via gradient information, that is, the derivative of the optimization functional with respect to the control parameters. Most generally, the control parameters are the amplitudes of time-dependent control fields, e.g., laser pulses in the control of trapped atoms or ions, and microwave pulses for the control of superconducting circuits.

Traditionally, the required gradients have to be derived analytically. In particular for GRAPE, the numerical scheme [12] is formulated specifically for an overlap with a target state. This limits existing GRAPE implementations [23–26] to a small class of optimization functionals that include state-to-state transfers and quantum gates. The implementation of Krotov's method [27] provides slightly

more flexibility, but still requires that a function to evaluate the gradient is passed along with the optimization functional.

The limitation to standard functionals [19] has restrained the full potential of quantum control. In many applications, both in quantum information and in quantum sensing, there are figures of merit that capture the true objective of the optimization but do not fit into the simple mold of reaching a specific target state. For example, the true intent of a control scheme is often to generate entanglement. Similarly, in metrology, the quantum Fisher information [28] directly measures the metrological gain [29, 30].

In the context of universal quantum computing, in combination with single-qubit gates, *any* perfectly entangling two-qubit gate is sufficient to implement a quantum circuit [8]. For a complex system, which specific gate can be implemented with minimal resources or with maximum noise robustness is not predictable, but can be identified by directly maximizing the entanglement power, as defined by the gate concurrence [31]. This is complicated by the fact that the gate concurrence is not an analytic quantity and thus there is no closed-form expression for the gradient. To address this, based on the geometric theory of two-qubit gates in the Weyl chamber [32, 33], an alternative functional that has an analytic, albeit complicated, gradient was formulated and demonstrated in Refs. [34, 35].

A breakthrough in the flexibility of quantum optimal control was made by adopting automatic differentiation (AD) in Refs. [36–40], with proof-of-concept implementations in Refs. [41, 42]. Automatic differentiation [43, 44] considers the evaluation of the optimization functional as a computational graph of elementary operations, and applies the chain rule to evaluate its derivative. This relies on the realization that at a sufficiently low level, *any* function numerically evaluated by a computer is analytic.

In addition to providing the flexibility to include arbitrary final-time functional and running costs, AD has enabled the optimization of open quantum systems through quantum trajectories [38, 45, 46], which are otherwise difficult to incorporate analytically [47]. The approach of gradient-based optimization through AD has also been taken up in some more comprehensive quantum control software packages [48–50].

Although first developed in the 1960s [51], the widespread use of AD is associated with the rise of machine learning. It is at the core of the backpropagation method [52] for training neural networks. Hence, it is most commonly embedded in machine learning frameworks such as Tensorflow [53], PyTorch [54, 55], JAX [56, 57], or Flux/Zygote [58–61]. These frameworks have traditionally focused only on the numerical operations required for neural networks, specifically dense real-valued matrix-vector multiplications. The in-place sparse complex-valued linear algebra operations required for the efficient numerical methods of quantum dynamics [62–66] have only recently started to be addressed and thus have required workarounds that have hindered performance.

More fundamentally, AD requires the storage of gradient-information for every node in the computational graph. In a "full-AD" mode as in Refs. [36–40], where the full time propagation as well as the evaluation of the optimization functional are performed within the AD framework, this generally implies the storage of a gradient matrix or vector for every linear algebra operation. For large Hilbert space dimensions, open quantum systems, or a large number of control parameters (time steps), this numerical overhead quickly becomes prohibitive.

In this paper, we address both of these issues by introducing the approach of semi-automatic differentiation. The approach exploits the fact that all pulse-level quantum control problems are based on the time evolution of quantum states. We show how formally rewriting the optimization functional as a function of the propagated states, of the overlaps with target states, or of a quantum gate allows to split the evaluation of the gradient into two parts. The main part can be evaluated analytically and leads to an efficient modified GRAPE scheme. The remaining part has a profoundly reduced computational complexity and can be evaluated using automatic differentiation with negligible numerical overhead.

The modified GRAPE scheme is described in detail and compared with Krotov's method. Thus, the paper gives a blueprint for the efficient implementation of gradient-based optimal control for arbitrary functionals. In addition, we have also implemented the method in the Julia programming

language [67, 68] in two packages `GRAPE.jl` [69] and `Krotov.jl` [70], both of which are part of a more comprehensive `QuantumControl.jl` framework [71].

To demonstrate the numerical efficacy of the semi-automatic differentiation approach, we benchmark the optimization of entangling quantum gates for two superconducting transmon qubits [72] with a shared transmission line [73] for a varying Hilbert space dimension and a varying number of time steps. In addition to replicating the results of Refs. [34, 35] with an automatic gradient, we also demonstrate the first *direct* optimization of a non-analytic entanglement measure [31]. We show that the numerical cost of the optimization in terms of both memory and runtime scales identically to the direct optimization of a specific quantum gate with an analytic gradient. This is in stark contrast to the use of full automatic differentiation along the lines of Refs. [36–40], which we show to quickly become infeasible in terms of memory usage and/or runtime.

The paper is structured as follows. In Section 2, we review the efficient implementation of GRAPE and Krotov's method. This includes the calculation of gradients to machine precision. The numerical scheme described here also applies to the analytic component of the semi-AD approach, with only minor changes. In Section 3, we briefly review the concepts of automatic differentiation to develop an understanding of the potential implications of a full-AD approach on numerical costs. Section 4 then develops the theory of semi-automatic differentiation and contains the main new results of this paper. Section 5 defines the optimization problem for perfectly entangling quantum gates on superconducting transmon qubits and shows the benchmarks of the semi-AD and full-AD approaches, as well as the direct optimization of a quantum gate. Section 6 concludes.

2 Gradient-based Optimal Control

In this section, we review standard methods of gradient-based numerical control theory. As we will show in section 4, the method of semi-automatic differentiation that we introduce in this paper builds on these existing control methods, with minimal additional numerical overhead. That is, the numerical cost of optimizing *arbitrary* functionals with semi-automatic differentiation is virtually the same as the cost of optimizing *standard* functionals with traditional gradient-based control methods. Thus, we review the state-of-the-art for implementing these methods as efficiently as possible.

2.1 Optimization Functionals

Mathematically, quantum control problems are solved by iteratively minimizing an optimization functional of the general form

$$J(\{\epsilon_{nl}\}) = J_T(\{|\Psi_k(T)\rangle\}) + \int_0^T g_a\big(\{\epsilon_l(t)\}, t\big)\,\mathrm{d}t + \int_0^T g_b\big(\{|\Psi_k(t)\rangle\}, t\big)\,\mathrm{d}t\,. \tag{1}$$

The terms that constitute the total functional are the final time functional J_T and the running costs $g_{a,b}(t)$ that may encode penalties on the pulse amplitudes or on the propagated states. J_T depends explicitly on the states $\{|\Psi_k(T)\rangle\}$. These are the result of a forward propagation of some set of states $\{|\phi_k\rangle\}$ at $t = 0$ under the control fields $\epsilon_l(t)$. The index l numbers independent control fields. In the example in Section 5, this would be the real and imaginary part of a complex-valued microwave field in a rotating frame, or equivalently the amplitude and phase of the microwave field in the non-rotating frame. The index k numbers different "objectives" that must be achieved simultaneously. For the example of a two-qubit gate $\hat{\mathrm{O}}$, k numbers the four logical basis states $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$. Then, a typical functional is

$$J_T = J_{T,\mathrm{sm}} = 1 - \Big|\frac{1}{N}\sum_{k=1}^{N}\langle\phi_k^{\mathrm{tgt}}|\hat{\mathrm{U}}|\phi_k\rangle\Big|^2, \tag{2}$$

with $N = 4$, where $|\phi_k^{\mathrm{tgt}}\rangle \equiv \hat{\mathrm{O}}\,|\phi_k\rangle$ is a target state at $t = T$ and $\hat{\mathrm{U}}$ is the time evolution operator, so that $|\Psi_k(T)\rangle \equiv \hat{\mathrm{U}}\,|\phi_k\rangle$.

On the left-hand side the functional J explicitly depends on a set of control parameters $\{\epsilon_{nl}\}$. We will primarily focus here on piecewise constant control fields. In this case, ϵ_{nl} is the value of the l'th control field on the n'th interval of the time grid. With N_T time intervals, the time evolution operator is then $\hat{\mathsf{U}} = \prod_{n=N_T}^{n=1} e^{-\mathrm{i}\hat{\mathsf{H}}_n dt_n}$ where $\hat{\mathsf{H}}_n$ is the Hamiltonian for the n'th interval on the time grid, $dt_n = t_n - t_{n-1}$, typically $\hat{\mathsf{H}}_n = \hat{\mathsf{H}}^{(0)} + \sum_l \epsilon_{nl}\hat{\mathsf{H}}^{(l)}$ with the drift Hamiltonian $\hat{\mathsf{H}}^{(0)}$ and the control Hamiltonians $\{\hat{\mathsf{H}}^{(l)}\}$; although a nonlinear dependency on the control fields is also possible.

While we have written the functional and the time evolution in terms of Hilbert space states and Hamiltonians, all of the formalism applies equally to open quantum systems. In this case, any state $|\Psi\rangle$ is replaced by a density matrix $\hat{\rho}$ and any Hamiltonian $\hat{\mathsf{H}}$ is replaced by a Liouvillian super-operator $\mathcal{L}$. Any inner product $\langle\Psi|\Phi\rangle$ is defined for (density) matrices as $\mathrm{tr}[\hat{\mathsf{A}}^\dagger\hat{\mathsf{B}}]$. In any case, from a numerical perspective, a state $|\Psi\rangle$ or a (vectorized) density matrix $\hat{\rho}$ is a complex vector, and a Hamiltonian $\hat{\mathsf{H}}$ or a Liouvillian $\mathcal{L}$ is a (sparse) matrix acting on that vector. The main distinction is that $\hat{\mathsf{H}}$ for a normal Schrödinger equation is Hermitian, while $\mathcal{L}$ is not. For open quantum systems described by a master equation in Lindblad form, the explicit $\mathcal{L}$ can be constructed as a sparse matrix from the Lindblad operators; see, e.g., Appendix B.2 in Ref. [74].

2.2 GRAPE

The most direct gradient-based method for minimizing a functional as in Eq. (1) is Gradient Ascent Pulse Engineering (GRAPE) [12]. In its original form, the optimization is performed by calculating the gradient ∇J for the piecewise-constant pulse values as the control parameters, i.e., the vector of values $\partial J/\partial\epsilon_{nl}^{(i)}$ for all values n, l of the control field in iteration (i), and then update the control field in the direction of the gradient as $\epsilon_{nl}^{(i+1)} = \epsilon_{nl}^{(i+1)} - \alpha(\nabla J)_{nl}$ with some fixed step width (or "learning rate") α.

In practice, once the gradient ∇J has been calculated, it can be fed into a black-box gradient-based optimization package, e.g. the Matlab or SciPy optimization toolboxes [75–77] or standalone packages such as `Optim.jl` [78]. For one, these packages will perform a linesearch to determine a suitable step width α in each iteration. Even more importantly, they can employ quasi-Newton methods for the optimization that use a Hessian (the matrix of second order derivatives) estimated only from the gradient information of previous iterations. Using second-order information in this way dramatically speeds up convergence [13] and is strongly recommended.

We find that L-BFGS-B [79], written in Fortran [80] with wrappers for Python [77] and Julia [81] is a particularly robust quasi-Newton optimizer. It also includes the possibility to apply bounds to the control field (hence the suffix -B). Its particular line search method requires recalculating ∇J for each step, so it is numerically more expensive than some other linesearch methods that use only *evaluations* of J to determine α. Other implementations of LBFGS [78] allow for a variety of linesearch algorithms [82] with more customization. However, we have not found any of these to reliably yield better convergence than the method built in to L-BFGS-B, and indeed have found the lack of hyperparameters in L-BFGS-B to be a virtue.

2.3 Efficient Evaluation of Final-Time Gradients

Typically, implementations of GRAPE only consider specific final-time functionals, $J \equiv J_T$ in Eq. (1), e.g., $J_T = J_{T,\mathrm{sm}}$ given by Eq. (2). We will discuss the efficient evaluation of the gradient for this special case here, and the more general case with non-zero running costs in Section 4.4. To evaluate ∇J_T, we define $\tau_k \equiv \langle\phi_k^{\mathrm{tgt}}|\Psi_k(T)\rangle$ with $|\Psi_k(T)\rangle = \hat{\mathsf{U}}\,|\phi_k\rangle = \hat{\mathsf{U}}_{NT}\ldots\hat{\mathsf{U}}_1\,|\phi_k\rangle$ for each of the k objectives. The time evolution operators $\hat{\mathsf{U}}_n$ for the time intervals $n = 1\ldots N_T$ are piecewise constant. We first consider

$$\nabla\tau_{nl}^{(k)} \equiv \frac{\partial}{\partial\epsilon_{nl}}\langle\phi_k^{\mathrm{tgt}}|\Psi_k(T)\rangle = \frac{\partial}{\partial\epsilon_{nl}}\langle\phi_k^{\mathrm{tgt}}|\hat{\mathsf{U}}_{N_T}\ldots\hat{\mathsf{U}}_n\ldots\hat{\mathsf{U}}_1|\phi_k\rangle = \left\langle\chi_k(t_n)\middle|\frac{\partial\hat{\mathsf{U}}_n}{\partial\epsilon_{nl}}\middle|\Psi_k(t_{n-1})\right\rangle \tag{3}$$

with $|\chi_k(t_n)\rangle = U_{n+1}^\dagger \dots U_{N_T}^\dagger |\phi_k^{\text{tgt}}\rangle$, i.e., a backward-propagation of the target state with the adjoint Hamiltonian or Liouvillian and $|\Psi_k(t_{n-1})\rangle = \hat{\mathsf{U}}_{n-1} \dots \hat{\mathsf{U}}_1 |\phi_k\rangle$, i.e., a forward-propagation of the initial state.

The derivative of the time evolution operator $\hat{\mathsf{U}}_n$ of a particular time step acting on an arbitrary state can be evaluated efficiently and to machine precision by defining a "gradient generator" for the n'th time step as a block matrix

$$G_n = \begin{pmatrix} \hat{\mathsf{H}}_n & 0 & \dots & 0 & \hat{\mathsf{H}}_n^{(1)} \\ 0 & \hat{\mathsf{H}}_n & \dots & 0 & \hat{\mathsf{H}}_n^{(2)} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & \hat{\mathsf{H}}_n & \hat{\mathsf{H}}_n^{(L)} \\ 0 & 0 & \dots & 0 & \hat{\mathsf{H}}_n \end{pmatrix} \tag{4}$$

where $\hat{\mathsf{H}}_n^{(l)} \equiv \frac{\partial \hat{\mathsf{H}}_n}{\partial \epsilon_{nl}}$ for the different controls numbered 1 through L, and $\hat{\mathsf{H}}_n$ is the full Hamiltonian or Liouvillian for that time step. It can be shown that [83, 84]

$$e^{-\mathrm{i}G_n dt_n} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ |\Psi\rangle \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial \epsilon_{n1}} e^{-\mathrm{i}\hat{\mathsf{H}}_n^{(1)} dt_n} |\Psi\rangle \\ \vdots \\ \frac{\partial}{\partial \epsilon_{nL}} e^{-\mathrm{i}\hat{\mathsf{H}}_n^{(L)} dt_n} |\Psi\rangle \\ e^{-\mathrm{i}\hat{\mathsf{H}}_n dt_n} |\Psi\rangle \end{pmatrix} = \begin{pmatrix} \frac{\partial \hat{\mathsf{U}}_n}{\partial \epsilon_{n1}} |\Psi\rangle \\ \vdots \\ \frac{\partial \hat{\mathsf{U}}_n}{\partial \epsilon_{nL}} |\Psi\rangle \\ \hat{\mathsf{U}}_n |\Psi\rangle \end{pmatrix} . \tag{5}$$

That is, by propagating an extended vector $|\tilde{\Psi}\rangle = [|\tilde{\psi}_1\rangle, \dots, |\tilde{\psi}_L\rangle, |\Psi\rangle]^T$ under G_n, where the components $|\tilde{\psi}_l\rangle$ for the L different controls are initialized to zero, we obtain both the forward propagation of an arbitrary state vector $|\Psi\rangle$ and the gradient of the time evolution operator for that step with respect to every control field. Note that the block matrix defined in Eq. (4) does not need to be instantiated; it is sufficient to define it as an abstract operator such that

$$G_n |\tilde{\Psi}\rangle = \begin{pmatrix} \hat{\mathsf{H}}_n |\tilde{\psi}_1\rangle + \hat{\mathsf{H}}_n^{(1)} |\Psi\rangle \\ \vdots \\ \hat{\mathsf{H}}_n |\tilde{\psi}_L\rangle + \hat{\mathsf{H}}_n^{(L)} |\Psi\rangle \\ \hat{\mathsf{H}}_n |\Psi\rangle \end{pmatrix} , \tag{6}$$

where $|\tilde{\psi}_l\rangle$ is the l'th block of the extended $|\tilde{\Psi}\rangle$ and $|\Psi\rangle$ is the state vector on which $|\tilde{\Psi}\rangle$ is based. Thus, the data structure encoding G_n can be essentially the same as the data structure encoding $\hat{\mathsf{H}}_n$.

The propagation defined in Eq. (5) can be performed using any propagation method used to evaluate the "normal" propagation $\hat{\mathsf{U}}_n |\Psi\rangle$; that is, a generic ODE solver, or preferably a more efficient polynomial expansion of the time evolution operator $\hat{\mathsf{U}}_n = e^{-\mathrm{i}\hat{\mathsf{H}}dt}$ for a uniform time step dt. For a Hermitian $\hat{\mathsf{H}}$, Chebychev polynomials are the fastest converging polynomial expansion [85]. For the standard Schrödinger equation, propagation with Chebychev polynomials $P_m(x)$ is very straightforward and efficient to implement. The time evolution of a state $|\Psi\rangle$ by a single time step dt for the n'th interval of the time grid can be written as

$$\hat{\mathsf{U}}_n |\Psi\rangle = \sum_m a_m P_m(-\mathrm{i}\hat{\mathsf{H}}_{n,\text{norm}}) |\Psi\rangle = \sum_m a_m |\Phi_m\rangle , \tag{7}$$

where $\hat{\mathsf{H}}_{n,\text{norm}}$ is the Hamiltonian $\hat{\mathsf{H}}_n$ normalized to a spectral range within $[-1, 1]$, and using the recursive definition of the Chebychev polynomials,

$$|\Phi_0\rangle = |\Psi\rangle , \tag{8a}$$

$$|\Phi_1\rangle = -\mathrm{i}\hat{\mathsf{H}}_{n,\text{norm}} |\Phi_0\rangle , \tag{8b}$$

$$|\Phi_m\rangle = -2\mathrm{i}\hat{\mathsf{H}}_{n,\text{norm}} |\Phi_{m-1}\rangle + |\Phi_{m-2}\rangle . \tag{8c}$$

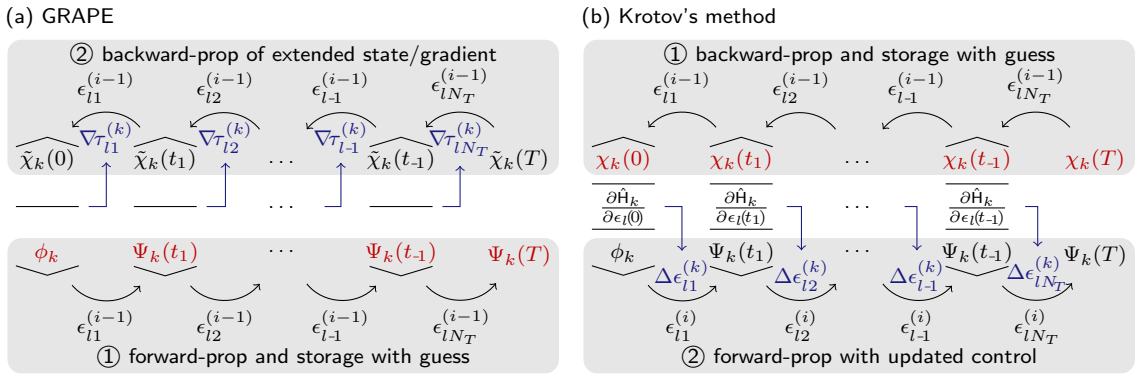

Figure 1: Numerical scheme for the contribution from the k'th objective to the gradient/update for the l'th control, in iteration (i) of an optimization with GRAPE (a) and Krotov's method (b). For GRAPE, the scheme starts in the bottom left with the initial state $|\phi_k\rangle$ at $t = 0$. The states marked in red must be stored in memory. The backward propagation of the extended state $|\tilde{\chi}_k(t)\rangle$ is defined in Eq. (9). Both the forward and the backward propagation uses the guess controls, indicated by the superscript $(i-1)$. A negative time index, e.g. in $t_{\text{-}1}$, is shorthand for $N_T - 1$. The gradient values $\nabla\tau_{nl}^{(k)}$ marked in blue are defined in Eq. (10). For Krotov's method, the schemes starts in the top right of panel (b) with $|\chi_k(T)\rangle$ defined in Eq. (14). The backward propagation uses the guess controls $(i-1)$, while the forward propagation uses the updated controls (i). The updates $\Delta\epsilon_{nl}^{(k)}$ marked in blue correspond to the terms under the sum in Eq. (12), at the midpoint of the n'th time interval. That is, they are summed over k to obtain the total updated $\epsilon_{nl}^{(i)}$.

The expansion coefficients a_m for a given spectral range and a uniform time step dt can be calculated analytically and are proportional to Bessel functions [62]. For a non-Hermitian $\hat{\mathsf{H}}$ (e.g., a Liouvillian), an expansion into Newton polynomials is suitable [64, 66, 86]. Both Chebychev and Newton propagation have been implemented in Julia [87].

In this context, it is worth noting that the eigenvalues of G_n in Eq. (4) are the same as the eigenvalues of the underlying $\hat{\mathsf{H}}_n$, with an additional $(L+1)$ degeneracy. In particular, for a Hermitian $\hat{\mathsf{H}}_n$ the eigenvalues of G_n are real, despite G_n as a whole not being Hermitian. Thus, if $e^{-\mathrm{i}\hat{\mathsf{H}}_n dt}$ can be evaluated via a Chebychev expansion, so can $e^{-\mathrm{i}G_n dt}$, with the same expansion coefficients.

The complete numerical scheme for evaluating $\nabla\tau_{nl}^{(k)}$ is shown in Fig. 1 (a). It starts at the bottom left with the initial state $|\phi_k\rangle$. This state is forward-propagated using the values $\epsilon_{nl}^{(i-1)}$ for the l'th control and the n'th time step, where the superscript $(i-1)$ indicates the guess for the current iteration (i). The propagation continues to the final state $|\Psi_k(T)\rangle$. All of these propagated states must be stored in memory. After the forward-propagation ends, we initialize an extended state $|\tilde{\chi}_k(t = t_{N_T} = T)\rangle = [0, \ldots 0, |\phi_k^{\text{tgt}}\rangle]^T$, consisting of a zero block for each of the L controls $\epsilon_l(t)$ and the target state for the objective (k) in the bottom block. This extended state is backward-propagated as $|\tilde{\chi}_k(t_{n-1})\rangle = e^{-\mathrm{i}G_n^* dt_n}\,|\tilde{\chi}_k(t_n)\rangle$ with a negative time step dt_n. In the full block-form of the extended state and generator, each propagation step is defined as

$$\begin{pmatrix} |\tilde{\chi}_{k1}(t_{n-1})\rangle \\ \vdots \\ |\tilde{\chi}_{kL}(t_{n-1})\rangle \\ |\chi_k(t_{n-1})\rangle \end{pmatrix} = \begin{pmatrix} \frac{\partial\hat{\mathsf{U}}_n^\dagger}{\partial\epsilon_{n1}}\,|\chi_k(t_n)\rangle \\ \vdots \\ \frac{\partial\hat{\mathsf{U}}_n^\dagger}{\partial\epsilon_{nL}}\,|\chi_k(t_n)\rangle \\ \hat{\mathsf{U}}_n^\dagger\,|\chi_k(t_n)\rangle \end{pmatrix} = \exp\left[-\mathrm{i}\begin{pmatrix} \hat{\mathsf{H}}_n^\dagger & 0 & \ldots & 0 & \hat{\mathsf{H}}_n^{(1)\dagger} \\ 0 & \hat{\mathsf{H}}_n^\dagger & \ldots & 0 & \hat{\mathsf{H}}_n^{(2)\dagger} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \ldots & \hat{\mathsf{H}}_n^\dagger & \hat{\mathsf{H}}_n^{(L)\dagger} \\ 0 & 0 & \ldots & 0 & \hat{\mathsf{H}}_n^\dagger\,, \end{pmatrix} dt_n\right] \begin{pmatrix} 0 \\ \vdots \\ 0 \\ |\chi_k(t_n)\rangle \end{pmatrix}, \quad (9)$$

which is the backward version of the forward step defined in Eqs. (4–6). After each step in the backward propagation, we calculate the nl'th component of the gradient of τ_k with respect to the control values,

$$\nabla\tau_{nl}^{(k)} \equiv (\nabla\tau_k)_{nl} = \langle\tilde{\chi}_{kl}(t_{n-1})\,|\,\Psi_k(t_{n-1})\rangle\,, \quad (10)$$

where $|\tilde{\chi}_{kl}(t_{n-1})\rangle$ is the state from the l'th block of $|\tilde{\chi}_k(t_{n-1})\rangle$ and $|\Psi_k(t_{n-1})\rangle$ is read from the stored forward-propagated states. After calculating this overlap, the first L blocks of $|\tilde{\chi}_k(t_{n-1}\rangle$ must be zeroed out before performing the next step in the backward propagation, so as to have the correct state on the right-hand side of Eq. (9).

The scheme depicted in Fig. 1 (a) is inherently parallel in the different objectives (index k). This is in fact one reason why we have framed the optimization of a quantum gate in terms of multiple objectives, one for each of the logical basis states. The approach is in contrast to the approach taken by most existing GRAPE implementations [23, 24, 37] that treat the quantum gate $\hat{\mathrm{U}}$ as the dynamic object and may even use explicit matrix-exponentiation to calculate $\hat{\mathrm{U}} = e^{-\mathrm{i}\hat{\mathrm{H}}dt}$, limiting the implementation to small Hilbert space dimensions. In addition to numerical parallelizability and efficiency, formulating the control problem in terms of multiple simultaneous objectives also allows the method to extend to gate optimization in open quantum systems [88] or to ensemble optimization for robust quantum gates [89].

In principle, we could reverse the order of the backward and forward propagation: Instead of the procedure shown in Fig. 1 (a), we could first backward-propagate $|\chi_k(T)\rangle = |\phi_k^{\mathrm{tgt}}\rangle$ and store the resulting $|\chi_k(t)\rangle$, and then forward-propagate an extended state $|\tilde{\Psi}_k(t)\rangle$ to evaluate $\nabla\tau_{nl}^{(k)}$, using Eqs. (4–6). This corresponds to $\frac{\partial\hat{\mathrm{U}}_n}{\partial\epsilon_{nl}}$ in Eq. (3) acting to the right instead of to the left. However, as we show in Section 4.1, doing the forward-propagation first is necessary when combining GRAPE with automatic differentiation.

Having evaluated $\nabla\tau_{nl}^{(k)}$ via Eq. (3), respectively Eq. (10), we find for the gradient of the square-modulus functional in Eq. (2)

$$(\nabla J_{T,\mathrm{sm}})_{nl} \equiv \frac{\partial J_{T,\mathrm{sm}}}{\partial\epsilon_{nl}} = -\frac{1}{N^2}\sum_{k,k'=1}^{N}\left[\frac{\partial\tau_{k'}^*}{\partial\epsilon_{nl}}\tau_k + \tau_{k'}^*\frac{\partial\tau_k}{\partial\epsilon_{nl}}\right] = -\frac{2}{N^2}\mathrm{Re}\sum_{k,k'=1}^{N}\tau_{k'}^*\nabla\tau_{nl}^{(k)}\,. \tag{11}$$

For other functionals [19], we would similarly have to compute the chain rule to complete the GRAPE scheme.

2.4 Krotov's method

As an alternative to GRAPE, Krotov's method [14–16] takes a constructive approach based on time-continuous control fields, $J(\{\epsilon_l(t)\})$ on the left-hand side of Eq. (1). For a specifically chosen running cost, $g_a(\epsilon_l(t)) = \frac{\lambda_a}{S(t)}[\epsilon_l(t) - \epsilon_l^{\mathrm{ref}}(t)]^2$, with an arbitrary "update shape" $S(t)$ and scaling factor λ_a, and a reference field $\epsilon_l^{\mathrm{ref}}(t)$ that is typically chosen in each iteration $(i+1)$ as the guess field $\epsilon_l^{(i)}(t)$ for that iteration, the derivation of Krotov's method [17–20] considers the necessary and sufficient conditions for the functional derivative $\frac{\partial J}{\partial\epsilon_l(t)}$ to ensure monotonic convergence, $J(\{\epsilon_l^{(i+1)}(t)\}) \leq J(\{\epsilon_l^{(i)}(t)\})$, and finds a first-order [20] update equation for $\Delta\epsilon_l^{(i)}(t) \equiv \epsilon_l^{(i+1)}(t) - \epsilon_l^{(i)}(t)$ in each iteration,

$$\Delta\epsilon_l^{(i)}(t) = \frac{S(t)}{\lambda_a}\mathrm{Im}\left[\sum_{k=1}^{N}\left\langle\chi_k^{(i-1)}(t)\right|\frac{\partial\hat{\mathrm{H}}}{\partial\epsilon_l^{(i)}(t)}\left|\Psi_k^{(i)}(t)\right\rangle\right]\,, \tag{12}$$

where $|\Psi_k(t)\rangle$ is the initial state $|\phi_k\rangle$ forward-propagated with the updated pulse for the current iteration, and $|\chi_k^{(i-1)}(t)\rangle$ is a state backward-propagated according to

$$\frac{\partial}{\partial t}|\chi_k^{(i-1)}(t)\rangle = -\mathrm{i}\hat{\mathrm{H}}^\dagger|\chi_k^{(i-1)}(t)\rangle + \frac{\partial g_b}{\partial\langle\Psi_k^{(i-1)}(t)|} \tag{13}$$

with the boundary condition

$$|\chi_k^{(i-1)}(T)\rangle = -\frac{\partial J_T}{\partial\langle\Psi_k^{(i-1)}(T)|}\,. \tag{14}$$

The discretization to a time grid is done only after formulating the time-continuous update equation and results in the scheme shown in Fig. 1 (b). For each objective indexed by k, we start in the top right with $|\chi_k(T)\rangle$ defined by Eq. (14). This state is backward-propagated as

$$|\chi_k(t_{n-1})\rangle = e^{-\mathrm{i}\hat{\mathsf{H}}_n^\dagger dt_n} |\chi_k(t_n)\rangle \tag{15}$$

($dt_n < 0$) if there are no state-dependent running costs ($g_b \equiv 0$), or using an inhomogeneous propagator otherwise [90]. All backward-propagated states must be stored in memory. For each control field indexed by l, the control update for the first time step is calculated according to Eq. (12) for $t = 0$, i.e., using the result of the backward propagation and the initial state $|\phi_k\rangle$ for each objective. The updated controls for the first time step then allow to obtain $|\Psi_k(t_1)\rangle$, which together with $|\chi_k(t_1)\rangle$ from the stored backward-propagated states allows to calculate the update for the second time step. In this sense, the scheme is *sequential*: the update in every time step depends directly on the state forward-propagated under the updated pulse from the previous time step. In contrast, the update in GRAPE is *concurrent*, with independent $\nabla\tau_{nl}^{(k)}$. Like the GRAPE scheme in panel (a), Krotov's method is inherently parallel in k, with the caveat that the parallelization must be synchronized after each time step in the forward propagation, so as to evaluate the sum over k in Eq. (12).

3 Automatic Differentiation

Existing implementations of GRAPE [23–26, 48] typically hard-code the optimization functional to something equivalent to Eq. (2). In order to extend to more functionals, at best a user has to manually supply a routine that calculates the gradient, for example, with a chain rule such as Eq. (11). In the worst case, if the gradient is not easily expressed as overlaps τ_k, the numerical scheme in Fig. 1 (a) would have to be adapted to that particular functional.

Even more problematic are figures of merit that are highly relevant to quantum engineering but do not have an analytic gradient. Typical examples are entanglement measures [31] or the quantum Fisher information [28] which is directly connected to metrological gain in quantum sensing applications [29, 30]. Evaluating these measures, in general, involves eigenvalue decompositions, which does not lend itself to an analytical calculation of the gradient. Even if equivalent analytic expressions, e.g., for the gate concurrence, can be found [34, 35], their analytic gradients are extremely tedious to calculate and implement [91].

In situations where the *analytic* calculation of a gradient is impossible or impractical, the *numerical* evaluation of gradients can be an alternative, in particular through automatic differentiation (AD) [36–40, 43, 44]. The core of AD is the realization that any numerical computation, even one that is seemingly non-analytical like an eigen-decomposition, can ultimately be expressed in elemental numerical steps – sums and products of floating point numbers, if taken to the extreme of machine instructions. These elemental steps have a known derivative, and thus the gradient of any function can be evaluated by applying the chain rule ad nauseam. Doing this requires that the computer keep track of all intermediate values in the computation. In our case, the function of interest is the optimization functional J, with input values $\{\epsilon_n\}$ (we temporarily drop the index l numbering different controls here for simplicity).

Consider

$$J(\epsilon_1, \epsilon_2) = \sin(\epsilon_1) + \epsilon_1\sqrt{\epsilon_2} \tag{16}$$

as a simple example, borrowed from Ref. [37]. We introduce intermediary values, $v_1 = \epsilon_1$, $v_2 = \epsilon_2$, $v_3 = \sin(v_1)$, $v_4 = \sqrt{v_2}$, $v_5 = v_1 v_4$, $v_6 = v_3 + v_5$, and thus finally $J = v_6$. The full chain rule for the gradient of J is

$$(\nabla J)_n \equiv \frac{\partial J}{\partial \epsilon_i} = \frac{\partial J}{\partial v_6}\frac{\partial v_6}{\partial v_3}\frac{\partial v_3}{\partial v_1}\frac{\partial v_1}{\partial \epsilon_n} + \frac{\partial J}{\partial v_6}\frac{\partial v_6}{\partial v_5}\frac{\partial v_5}{\partial v_1}\frac{\partial v_1}{\partial \epsilon_n} + \frac{\partial J}{\partial v_6}\frac{\partial v_6}{\partial v_5}\frac{\partial v_5}{\partial v_4}\frac{\partial v_4}{\partial v_2}\frac{\partial v_2}{\partial \epsilon_n}\,. \tag{17}$$

There are two ways to write the chain rule recursively, allowing to consistently evaluate it numerically for arbitrarily complicated functionals. These correspond to the *forward* and *reverse* modes of automatic differentiation.

For the forward mode, we define the "tangent" of an intermediary value as $\dot{v}_j \equiv \partial v_j / \partial \epsilon_n$ where we have picked a particular ϵ_n for which we want to evaluate the derivative. We then find

$$\dot{v}_j = \sum_i \frac{\partial v_j}{\partial v_i} \dot{v}_i \,, \tag{18}$$

where the sum is over all v_i on which v_j depends explicitly. We can evaluate these tangents along with the evaluation of the intermediary values themselves. For the derivative with respect to ϵ_1, in our example, we would have $\dot{v}_1 = 1$, $\dot{v}_2 = 0$, $\dot{v}_3 = \cos(\epsilon_1)$, $\dot{v}_4 = 0$, $\dot{v}_5 = \sqrt{\epsilon_2}$, $\dot{v}_6 = \sqrt{\epsilon_2} + \cos(\epsilon_1)$, which is indeed $\partial J / \partial \epsilon_1$. To calculate the full gradient, the entire calculation must be evaluated once for each ϵ_n. Thus, calculating the gradient in forward-mode is efficient only if the number of dependent variables is small. On the other hand, for a single dependent variable, the runtime and memory requirements of evaluating the derivative are proportional to those of evaluating J itself. In particular, a tangent $\dot{v}_j$ does not have to be kept in memory longer than the value v_j itself. This makes forward mode automatic differentiation easy to implement, e.g., by using operator overloading and dual numbers [43].

For the reverse mode, we define the "adjoint" of an intermediary value as $\bar{v}_j \equiv \partial J / \partial v_j$. The name "adjoint" in this context is unrelated to the Hermitian conjugate in quantum mechanics, denoted by a dagger. The definition of the AD adjoint results in a recursive relationship

$$\bar{v}_j = \sum_i \bar{v}_i \frac{\partial v_i}{v_j} \,, \tag{19}$$

where the sum is over all v_i which depend on v_j. Note that this is the reverse of Eq. (18). Consequently, the adjoints are evaluated backward, starting from $\bar{v}_6 = \partial J / \partial v_6 = 1$. We then further find $\bar{v}_5 = 1$, $\bar{v}_4 = \epsilon_1$, $\bar{v}_3 = 1$, $\bar{v}_2 = \epsilon_1 / (2\sqrt{\epsilon_2})$ and $\bar{v}_1 = \cos(\epsilon_1) + \sqrt{\epsilon_2}$. The final adjoints now contain the derivatives for all of the input parameters. This is the primary benefit of reverse-mode AD: the full gradient can be evaluated at once, making it the preferred method for calculating the gradient of an optimization functional $\mathbb{R}^{N_T L} \to \mathbb{R}$. However, we can only start the calculation of the adjoints once J itself has been evaluated. All intermediary values v_j must be stored together with information on which elemental function was used to obtain the value, as well as all adjoints $\bar{v}_j$.

In practice, this can be implemented in several ways. Early versions of Tensorflow [53] require that the entire calculation is set up as a computational graph, see Fig. 1 in Ref. [37]. A forward-pass through the graph calculates the functional and stores intermediary values, while a backward-pass distributes adjoint information to the parents of each node in the graph. Alternatively, a tabular representation of the graph may be constructed during the forward pass, in what is called a Wengert tape. The backward pass then adds adjoint information to the tape. Lastly, it is possible to transform the source code of a function that evaluates the optimization functional into a new function that first performs the original evaluation, and then inverts the computational steps, splicing in code to calculate the adjoints. These implementation details can have a significant impact on the performance and flexibility; see [92] for a discussion of the particular tradeoffs. However, they do not change the fundamental memory requirements associated with reverse-mode AD.

There is considerable flexibility in what is considered an "elemental" function for which the adjoint can be defined. Most importantly, linear algebra operations do not have to be handled at the level of scalar operations, alleviating to some extent the overhead associated with AD. Using the rules of matrix calculus [93, 94], AD adjoints for many operations can be defined [95]. This even includes eigen- or SVD decompositions [96].

The more high-level the elemental functions are, the less the numerical overhead of AD. However, this comes at the cost of having to define more and more analytical adjoints. This is why many AD frameworks have been slow to adopt operations that are outside of the narrow scope of machine learning, which only requires real-valued dense matrix-vector operations. In contrast, quantum dynamics

is inherently described with complex-valued state vectors, and operators are usually sparse. Defining AD adjoints for complex linear algebra operations is possible [97], but has only recently seen adoption. Another practical issue is that naively, the intermediary values (or vectors) v_i are immutable. Thus, most AD frameworks (including Zygote [60, 61], which we have used here) do not support in-place linear algebra (BLAS [98]) operations, which can greatly speed up the simulation of quantum dynamics.

4 Semi-Automatic Differentiation

We now develop a method to eliminate the two shortcomings of reverse-mode automatic differentiation: the excessive memory overhead associated with having to store the full computational graph, respectively, a Wengert tape, and the limited support in AD frameworks for the linear algebra operations relevant to simulating the dynamics of a quantum system. We do this by applying an analytic chain rule to the calculation of the gradient. To this end, we introduce intermediary variables z_j and rewrite the functional in terms of these intermediaries, $J(\{\epsilon_{nl}\}) \to J(\{z_j(\{\epsilon_{nl}\})\})$. The values z_j may be complex, which requires some care when writing out the chain rule.

In principle, one must separate the z_j into real and imaginary part as independent variables, $J = J(\{\mathrm{Re}[z_j]\}, \{\mathrm{Im}[z_j]\})$, resulting in

$$(\nabla J)_{nl} \equiv \frac{\partial J}{\partial \epsilon_{nl}} = \sum_j \left(\frac{\partial J}{\partial \mathrm{Re}[z_j]} \frac{\partial \mathrm{Re}[z_j]}{\partial \epsilon_{nl}} + \frac{\partial J}{\partial \mathrm{Im}[z_j]} \frac{\partial \mathrm{Im}[z_j]}{\partial \epsilon_{nl}} \right) . \tag{20}$$

An elegant alternative is to introduce Wirtinger derivatives,

$$\frac{\partial J}{\partial z_j} \equiv \frac{1}{2} \left(\frac{\partial J}{\partial \mathrm{Re}[z_j]} - \mathrm{i} \frac{\partial J}{\partial \mathrm{Im}[z_j]} \right) , \tag{21}$$

$$\frac{\partial J}{\partial z_j^*} \equiv \frac{1}{2} \left(\frac{\partial J}{\partial \mathrm{Re}[z_j]} + \mathrm{i} \frac{\partial J}{\partial \mathrm{Im}[z_j]} \right) = \left(\frac{\partial J}{\partial z_j} \right)^* , \tag{22}$$

which instead treats z_j and the conjugate value z_j^* as independent variables, so that

$$\frac{\partial J}{\partial \epsilon_{nl}} = \sum_j \left(\frac{\partial J}{\partial z_j} \frac{\partial z_j}{\partial \epsilon_{nl}} + \frac{\partial J}{\partial z_j^*} \frac{\partial z_j^*}{\partial \epsilon_{nl}} \right) = 2\mathrm{Re} \sum_j \frac{\partial J}{\partial z_j} \frac{\partial z_j}{\partial \epsilon_{nl}} . \tag{23}$$

The derivative of the complex value z_j with respect to the real value ϵ_{nl} is defined straightforwardly as

$$\frac{\partial z_j}{\partial \epsilon_{nl}} \equiv \frac{\partial \mathrm{Re}[z_j]}{\partial \epsilon_{nl}} + \mathrm{i} \frac{\partial \mathrm{Im}[z_j]}{\partial \epsilon_{nl}} . \tag{24}$$

Our goal is to choose parameters z_j so that $\partial J / \partial z_j$ can be calculated with automatic differentiation with minimal numerical effort. That is, we would like the computational graph for $J(\{z_j\})$ to be as small as possible. Additionally, if the number of parameters $\{z_j\}$ can be kept small, forward-mode differentiation or even the use of finite difference may become feasible. For the second part of the chain rule, we require that $\partial z_j / \partial \epsilon_{nl}$ can be calculated analytically (without the use of AD).

Software frameworks for automatic differentiation such as Zygote [61] and Tensorflow [53] may define a (mathematically questionable [94]) "gradient" of a real-valued function J with respect to a complex vector with elements z_j as

$$(\nabla_z J)_j \equiv \frac{\partial J}{\partial \mathrm{Re}[z_j]} + \mathrm{i} \frac{\partial J}{\partial \mathrm{Im}[z_j]} . \tag{25}$$

This differs from the Wirtinger derivatives by a factor of two. Thus,

$$\frac{\partial J}{\partial z_j} = \frac{1}{2} (\nabla_z J)_j^* \tag{26}$$

in Eq. (23) when using, e.g., Zygote's `gradient` function for $\nabla_z J$.

4.1 State functionals

As a starting point, we can take seriously the explicit dependency of J_T on $\{|\Psi_k(T)\rangle\}$ in Eq. (1) and write the gradient of J_T using the chain rule in the states. To do this, we must combine the Wirtinger derivative with the rules of matrix calculus [93, 94], and write

$$(\nabla J_T)_{nl} \equiv \frac{\partial J_T}{\partial \epsilon_{nl}} = 2\mathrm{Re} \sum_k \left(\frac{\partial J_T}{\partial |\Psi_k(T)\rangle} \frac{\partial |\Psi_k(T)\rangle}{\partial \epsilon_{nl}} \right) . \tag{27}$$

With the definition in Eq. (21), this corresponds directly to the scalar

$$\frac{\partial J_T}{\partial \epsilon_{nl}} = \sum_{km} \left(\frac{\partial J_T}{\partial \mathrm{Re}[\Psi_{km}]} \frac{\partial \mathrm{Re}[\Psi_{km}]}{\partial \epsilon_{nl}} + \frac{\partial J_T}{\partial \mathrm{Im}[\Psi_{km}]} \frac{\partial \mathrm{Im}[\Psi_{km}]}{\partial \epsilon_{nl}} \right) , \tag{28}$$

where $\Psi_{km} = \langle m | \Psi_k(T)\rangle$ for any orthonormal basis $\{|m\rangle\}$ corresponds to the z_j in Eq. (20).

In Eq. (27), we may now recognize that the derivative of the scalar J_T with respect to a column vector $|\Psi_k(T)\rangle$ results, according to the rules of matrix calculus, in a row vector that we may associate with a co-state $\langle \chi_k|$. Specifically, we may define

$$|\chi_k(T)\rangle \equiv -\frac{\partial J_T}{\partial \langle \Psi_k(T)|} \quad \Leftrightarrow \quad \langle \chi_k(T)| \equiv -\frac{\partial J_T}{\partial |\Psi_k(T)\rangle} , \tag{29}$$

with a minus sign that will be motivated in Section 4.5. Since $|\chi_k(T)\rangle$ does not depend on ϵ_{nl}, we may push it into the derivative and obtain

$$\frac{\partial J_T}{\partial \epsilon_{nl}} = -2\mathrm{Re} \sum_k \frac{\partial}{\partial \epsilon_{nl}} \langle \chi_k(T) | \Psi_k(T)\rangle . \tag{30}$$

We can now see that the term under the sum has the exact same form as Eq. (3), that is, the derivative of a complex overlap of two states, $\tau_k \equiv \langle \chi_k(T) | \Psi_k(T)\rangle$, which we know how to evaluate numerically via Eq. (10), respectively the scheme in Fig. 1 (a). The only difference is that in the original GRAPE, we initialize the extended state $|\tilde{\chi}_k(T)\rangle$ for the backward propagation from the target state, whereas now we initialize it with Eq. (29). This also explains why we have chosen to perform the forward-propagation first in Fig. 1 (a): while in the original GRAPE, backward and forward propagation are interchangeable, now we need the result $|\Psi_k(T)\rangle$ of the forward propagation in order to initialize the backward propagation.

We can use automatic differentiation to evaluate Eq. (29) for arbitrary functionals. For example, with Zygote's `gradient` function to evaluate $\nabla_{\Psi_k} J_T$ analogously to Eq. (25), we have

$$|\chi_k(T)\rangle = -\frac{1}{2} \nabla_{\Psi_k} J_T , \tag{31}$$

where the factor $\frac{1}{2}$ accounts for the difference between the complex gradient and the correct Wirtinger derivative, cf. Eq. (26).

4.2 Overlap functionals

When formulating the gradient for the square-modulus functional in Eq. (11), we already used the overlap of the forward-propagated state for the k'th objective with the target state for that objective, $\tau_k \equiv \langle \phi_k^{\mathrm{tgt}} | \Psi_k(T)\rangle$. Many of the most common functionals in quantum control are expressed in terms of overlaps of propagated states with target states [19]. We can exploit this to further analytically simplify the calculation of $|\chi_k(T)\rangle$ in Eq. (29) via automatic differentiation. We find

$$|\chi_k(T)\rangle = -\frac{\partial J_T}{\partial \langle \Psi_k(T)|} = -\left(\frac{\partial J_T}{\partial \tau_k^*} \frac{\partial \tau_k^*}{\partial \langle \Psi_k(T)|} \right) = -\frac{1}{2} \left(\nabla_{\tau_k} J_T \right) |\phi_k^{\mathrm{tgt}}\rangle , \tag{32}$$

where we have used that only the complex conjugate $\tau_k^* = \left\langle \Psi_k(T) \middle| \phi_k^{\text{tgt}} \right\rangle$ of the overlap depends explicitly on the co-state $\langle \Psi_k(T)|$. The gradient $\nabla_{\tau_k} J_T$ defined as in Eq. (26) can be obtained with automatic differentiation.

We note that for functionals that explicitly depend on overlaps [19], it is generally not difficult to evaluate the chain rule analytically. Thus, the use of automatic differentiation here is less a matter of necessity than of convenience. It allows us to implement a GRAPE optimization package where the user can pass an arbitrary functional $J_T(\{\tau_k\})$ without having to explicitly specify a gradient.

4.3 Gate functionals

For the optimization of quantum gates, such as the examples we will explore in Section 5, it is common to have a logical subspace embedded in a larger physical subspace. The functional J_T in this case can often be written as a function of the achieved gate $\hat{\mathrm{U}}_\mathrm{L}$ in the logical subspace.

In this context, $\hat{\mathrm{U}}_\mathrm{L}$ is the projection of the full time evolution operator $\hat{\mathrm{U}}$ to the logical subspace. Specifically, the entries of the matrix $\hat{\mathrm{U}}_\mathrm{L}$ are

$$(\hat{\mathrm{U}}_\mathrm{L})_{ij} = \langle \phi_i \,|\, \Psi_j(T) \rangle \quad \Leftrightarrow \quad (\hat{\mathrm{U}}_\mathrm{L})^*_{ij} = \langle \Psi_j(T) \,|\, \phi_i \rangle \;, \tag{33}$$

where $\{|\phi_i\rangle\}$ are the basis states that span the logical subspace (assumed to be the initial states for the optimization objectives), and each $|\Psi_j(T)\rangle$ is the result of forward-propagating $|\phi_j\rangle$.

We may then calculate the gradient of J_T as in Eqs. (29, 30), with a further analytic chain rule, just as in Section 4.2 but with the elements $(U_\mathrm{L})_{ij}$ instead of the complex overlaps τ_k:

$$|\chi_k\rangle \equiv -\frac{\partial J_T}{\partial \langle \Psi_k(T)|} = -\sum_{ij} \frac{\partial J_T}{\partial (U_\mathrm{L})^*_{ij}} \frac{\partial (U_\mathrm{L})^*_{ij}}{\partial \langle \Psi_k(T)|} \,, \tag{34}$$

again using the notation of the Wirtinger derivative. We have used that only $(U_\mathrm{L})^*_{ij}$ depends explicitly on the co-states $\{\langle \Psi_k(T)|\}$. Furthermore,

$$\frac{\partial J_T}{\partial (U_\mathrm{L})^*_{ij}} = \frac{1}{2} (\nabla_{U_\mathrm{L}} J_T)_{ij} \tag{35}$$

according to the definitions in Eqs. (25, 22), and

$$\frac{\partial (U_\mathrm{L})^*_{ij}}{\partial \langle \Psi_k(T)|} = \frac{\partial}{\partial \langle \Psi_k(T)|} \langle \Psi_j(T) \,|\, \phi_i \rangle = \delta_{jk} \, |\phi_i\rangle \tag{36}$$

with the Kronecker delta δ_{jk}. Thus, Eq. (34) simplifies to

$$|\chi_k\rangle = -\frac{1}{2} \sum_i (\nabla_{U_\mathrm{L}} J_T)_{ik} \, |\phi_i\rangle \;, \tag{37}$$

where $\nabla_{U_\mathrm{L}} J_T$ is evaluated via automatic differentiation, e.g. with Zygote's `gradient` function.

While the simplified Eqs. (32, 37) are not *fundamentally* different from constructing the boundary states $\{|\chi_k\rangle\}$ directly with automatic differentiation, they can help to circumventing numerical instabilities that an AD framework may have for complicated functionals. Also, for very large Hilbert space dimensions, it may eliminate some of the numerical overhead associated with automatic differentiation. While the direct construction of $|\chi_k\rangle$ requires differentiation in a number of variables proportional to the full Hilbert space dimension, using Eq. (32) reduces this to the dimension of the logical subspace, 4 complex numbers in the case of a two-qubit gate. Similarly, for Eq. (37) we have a reduction to the square of the dimension of the logical subspace, that is, 16 complex numbers for a two-qubit gate.

4.4 Running costs

So far, we have only discussed the evaluation of gradients for final time functionals J_T. We now extend the discussion of semi-automatic differentiation to the running costs $g_{a,b}$ in Eq. (1). Since we are considering piecewise constant pulses, the integral over the running cost turns into a sum over the time steps. That is, we rewrite Eq. (1) as

$$J(\{\epsilon_{nl}\}) = J_T(\{|\Psi_k(T)\rangle\}) + \sum_{n=1}^{N_T}\sum_l (g_a)_{nl} + \sum_{n=0}^{N_T}(g_b)_n\,, \tag{38}$$

with

$$(g_a)_{nl} = \frac{1}{dt_n} g_a(\epsilon_{nl}, dt_n)\,, \qquad (g_b)_n = \frac{1}{\Delta t_n} g_b(\{|\Psi_k(t_n)\rangle\}, t_n)\,. \tag{39}$$

As in Fig. 1, we define $|\Psi_k(t_n)\rangle = \hat{\mathrm{U}}_n \ldots \hat{\mathrm{U}}_1 |\phi_k\rangle$, $t_0 = 0$, $t_{N_T} = T$, and $\hat{\mathrm{U}}_n = \exp[-i\hat{\mathrm{H}}_n dt_n]$ as the time evolution operator for the n'th time interval, $dt_n = t_n - t_{n-1}$. Similarly, Δt_n is the time step around the time grid point t_n, e.g. $\Delta t_0 = dt_1$, $\Delta t_n = \frac{1}{2}(t_{n+1} - t_{n-1})$ for $1 \leq n < N_T$, and $\Delta t_{N_T} = dt_{N_T}$. For uniform time grids, $dt_n \equiv \Delta t_n \equiv dt$.

Typically, running costs on the control fields are direct analytic expressions, e.g., $g_a(\{\epsilon_{nl}\}) = \lambda_a \epsilon_{nl}^2$ to penalize large amplitudes, with a weight λ_a. Thus, they are easily included in the gradient, e.g., $(\nabla g_a)_{nl} = 2\lambda_a \epsilon_{nl}$. For convenience, this can also be done with automatic differentiation. This even extends to penalties on the first and second derivatives of the controls [37–39].

More interesting is the case of state-dependent constraints. Typical examples [99] include trajectory optimizations,

$$g_{b,\mathrm{trj}}(\{|\Psi_k(t_n)\rangle\}) = \lambda_b \sum_k \left\| |\Psi_k(t_n)\rangle - |\Psi_k^{\mathrm{tgt}}(t_n)\rangle \right\|^2\,, \tag{40}$$

where the time evolution of each state $|\Psi_k(t_n)\rangle$ should be close to some target evolution $|\Psi_k^{\mathrm{tgt}}(t_n)\rangle$ with a weight λ_b, or observable optimizations

$$g_{b,\hat{\mathrm{D}}(t)}(\{|\Psi_k(t_n)\rangle\}) = \lambda_b \sum_k \left\langle \Psi_k(t_n) \middle| \hat{\mathrm{D}}(t_n) \middle| \Psi_k(t_n) \right\rangle\,, \tag{41}$$

where the expectation value of some observable $\hat{\mathrm{D}}(t)$ is to be minimized. A special case of this is the minimization of the population in some forbidden subspace [100], where $\hat{\mathrm{D}}(t_n) \equiv \hat{\mathrm{D}}$ is a projector into that subspace.

To obtain the full gradient of a functional with a state-dependent running cost, we apply the same procedure as in Section 4.1 and find

$$\frac{\partial J}{\partial \epsilon_{nl}} = 2\mathrm{Re}\sum_k \left[\frac{\partial J_T}{\partial |\Psi_k(T)\rangle} \frac{\partial |\Psi_k(T)\rangle}{\partial \epsilon_{nl}} + \sum_{n'=0}^{N_T} \frac{\partial (g_b)_{n'}}{\partial |\Psi_k(t_{n'})\rangle} \frac{\partial |\Psi_k(t_{n'})\rangle}{\partial \epsilon_{nl}} \right] \tag{42a}$$

$$= -2\mathrm{Re}\sum_k \frac{\partial}{\partial \epsilon_{nl}} \left[\left\langle \chi_k^{(T)} \middle| \hat{\mathrm{U}}_{N_T} \ldots \hat{\mathrm{U}}_1 \middle| \phi_k \right\rangle + \sum_{n'=n}^{N_T} \left\langle \xi_k(t_{n'}) \middle| \hat{\mathrm{U}}_{n'} \ldots \hat{\mathrm{U}}_1 \middle| \phi_k \right\rangle \right] \tag{42b}$$

with

$$|\chi_k^{(T)}\rangle \equiv -\frac{\partial J_T}{\partial \langle \Psi_k(T)|}\,, \qquad |\xi_k(t_{n'})\rangle \equiv -\frac{\partial (g_b)_{n'}}{\partial \langle \Psi_k(t_{n'})|}\,, \tag{43}$$

cf. Eqs. (14, 29). In the sum over n' in Eq. (42b), we have used that $|\Psi_k(t_{n'})\rangle$ depends on ϵ_{nl} only for $n' \geq n$. This implies that for the final time interval, $n = N_T$, there is only a single term,

$$\frac{\partial J}{\partial \epsilon_{N_T l}} = -2\mathrm{Re}\sum_k \left\langle \chi_k(T) \middle| \frac{\partial \hat{\mathrm{U}}_{N_T}}{\partial \epsilon_{N_T l}} \middle| \Psi_k(t_{N_T - 1}) \right\rangle\,, \tag{44}$$

with

$$|\chi_k(T)\rangle \equiv |\chi_k^{(T)}\rangle + |\xi_k(T)\rangle = -\left(\frac{\partial J_T}{\partial \langle\Psi_k(T)|} + \frac{\partial (g_b)_{N_T}}{\partial \langle\Psi_k(T)|}\right). \tag{45}$$

Evaluating the gradient progressively backward in time for $n = (N_T - 1)\ldots 1$, we then find a recursive relationship

$$\frac{\partial J}{\partial \epsilon_{nl}} = -2\mathrm{Re}\sum_k \left\langle \chi_k(t_n)\middle|\frac{\partial \hat{\mathrm{U}}_n}{\partial \epsilon_{nl}}\middle|\Psi_k(t_{n-1})\right\rangle, \tag{46}$$

with

$$|\chi_k(t_n)\rangle = \hat{\mathrm{U}}_{n+1}^\dagger\,|\chi_k(t_{n+1})\rangle - \frac{\partial (g_b)_n}{\partial \langle\Psi_k(t_n)|}. \tag{47}$$

Thus, there are no fundamental changes to the scheme in Fig. 1 (a) in the presence of state-dependent running costs. The states $\{|\phi_k\rangle\}$ must be forward-propagated and stored, and then the extended states $|\tilde{\chi}_k(t_n)\rangle$ are propagated backward to produce the gradient. The only difference is that the boundary state $|\tilde{\chi}_k(T)\rangle$ is now constructed based on Eq. (45) instead of Eq. (29) (or the target states, in the original GRAPE). Furthermore, the backward-propagation uses the discrete inhomogeneous Eq. (47). The inhomogeneity is calculated using the forward-propagated states stored previously, with the derivative of g_b performed analytically or by automatic differentiation.

4.5 Semi-AD for Krotov's method and time-continuous schemes

We have included a minus sign in the definition of $|\chi_k(T)\rangle$, Eq. (29), respectively $|\chi_k^{(T)}\rangle$ in Eq. (43), since that results in the exact same definition as the $|\chi_k(T)\rangle$ that is the boundary condition for the backward propagation in Krotov's method, Eq. (14). This allows us to make a strong connection between the most general case of semi-automatic differentiation for GRAPE and for Krotov's method.

Evaluating Eq. (31) with a framework for automatic differentiation like Zygote or Tensorflow is all that is required to bring the concept of semi-automatic differentiation to Krotov's method. In this way, it becomes possible to use Krotov's method to optimize towards any computable functional. Conversely, for functionals where the derivative with respect to the states is known analytically, e.g., because they have already been explored using Krotov's method, that existing code can be shared between an implementation of GRAPE and Krotov's method.

We may also observe that the generalization of GRAPE and Krotov's method are even more closely related in the time-continuous limit. For $dt \to 0$, we may use the first-order Taylor expansion of $e^{-\mathrm{i}\hat{\mathrm{H}}_n dt}$ to find

$$\nabla J_T \approx -2\,dt\,\mathrm{Im}\left[\sum_{k=1}^{N}\left\langle \chi_k^{(i-1)}(t)\middle|\frac{\partial \hat{\mathrm{H}}}{\partial \epsilon_l(t)}\middle|\Psi_k^{(i-1)}(t)\right\rangle\right], \tag{48}$$

with the boundary condition of Eq. (29) for $|\chi_k(T)\rangle$. With $\Delta\epsilon(t) \propto -\nabla J_T$, this matches Krotov's update equation (12), up to the concurrent $|\Psi_k^{(i-1)}(t)\rangle$ in Eq. (48) replacing the sequential $|\Psi_k^{(i)}(t)\rangle$ in Eq. (12).

Similarly, if there are state-dependent constraints $g_b \not\equiv 0$ in Eq. (1), the time-continuous limit of Eq. (46) for the backward propagation in the semi-AD GRAPE method corresponds directly to the inhomogeneous Eq. (13) for the backward-propagation in Krotov's method. However, the former does not require a genuine inhomogeneous propagator [90]: the operator $\hat{\mathrm{U}}_{n+1}^\dagger$ in Eq. (46) is a normal time evolution operator, with the inhomogeneity added afterward.

Equation (48) may provide an alternative implementation of a generalized GRAPE scheme in the limit of very small dt. The scheme would be nearly identical to Fig. 1 (b); instead of calculating the pulse update directly, it would calculate the gradient ∇J_T and feed it into the L-BFGS-B optimizer. Considering the time evolution operator only to first order in dt avoids having to construct and propagate the extended state discussed in Section 2.3 and may thus be slightly faster. Finally, Eq. (48) may also provide a motivation for the exploration of optimization schemes that mix and match sequential and concurrent updates [24, 99].

4.6 Asymptotic memory usage and checkpointing

We can now analyze the asymptotic memory usage of the semi-automatic differentiation procedure in relation to the "full" use of automatic differentiation as in Refs. [36–40] in general terms. As discussed in Section 3, automatic differentiation in general requires the storage of any intermediate value in the evaluation of the functional, that is, the time propagation of the quantum states. Thus, the AD memory overhead depends very much on how exactly this propagation is implemented.

In many of the existing uses of AD for quantum control, the time propagation is achieved by exponentiating the Hamiltonian. The required storage if this exponentiation is performed in a single computational step has been analyzed in Ref. [101]. The intermediary values in this case are the time evolution operators, i.e., the exponentiated matrices, and the states resulting from the application of those time evolution operators, for every time step. For a Hilbert space dimension of N_H, the size of the matrices is N_H^2. The required storage for these matrices is thus proportional to $N_T N_H^2$ where N_T is the number of time steps. Asymptotically, this dominates over the storage required for the states, which is proportional to $N_T N_H$.

While matrix exponentiation in a single step minimizes the number of intermediary values in the computation graph and thus the AD memory overhead, the computational complexity for algorithms for matrix exponentiation scales polynomially in matrix-matrix multiplications, which themselves scale quite unfavorably as N_H^3. In contrast, expanding the time evolution operator in a polynomial series and applying it directly to the states, e.g., using the Chebychev propagation method outlined in Section 2.3 reduces the computational complexity to something polynomial in matrix-vector products, which scale as N_H^2. However, in the context of AD, the polynomial expansion also increases the number of intermediary states. Specifically, for a polynomial of order M, there are M intermediary matrices and M intermediary states, and thus the total asymptotic memory overhead increases by a factor of M. Typically, the M required for convergence of the polynomial is between 10 and 100, depending on the spectral radius of the Hamiltonian and the size of the time step.

A well-established technique to reduce the excessive memory overhead of automatic differentiation by trading it for an increase in runtime is the use of checkpointing [44]. The central idea of checkpointing is to store only periodic snapshots of the intermediate variables in the computation graph, and then recalculate the forward pass from the last available checkpoint when calculating the gradient via automatic differentiation. This idea has been applied to the use of automatic differentiation in GRAPE [101]. A snapshot is taken every $C = \sqrt{N_T}$ time steps in the propagation. The asymptotic memory usage in this case reduces from $N_T N_H^2$ to $(N_C + C)N_H^2 = 2\sqrt{N_T} N_H^2$, where $N_C = N_T/C$ is the number of checkpoints. That is, checkpointing achieves a quadratic reduction with respect to the number of time steps. With a polynomial propagator, we again have an increase by a factor of M within each checkpointed segment of size C, thus resulting in a memory scaling of $(N_C + MC)N_H^2 \approx M\sqrt{N_T} N_H^2$, again a quadratic reduction.

The fundamental idea of checkpointing can also be applied in the semi-AD scheme in Fig. 1 (a). By default, we store all the forward propagated states marked in red. Instead, we may store only every C states. During the backward propagation of $|\tilde{\chi}_k(T)\rangle$, we then repeat the forward propagation from the last available checkpoint to recover that states $|\Psi_k(t_n)\rangle$ required for the overlaps that determine $\nabla\tau_{ln}^{(k)}$. The required memory for storage is thus reduced from N_T to $N_C + C = 2\sqrt{N_T}$. However, we need an additional $N_T - N_C = N_T - \sqrt{N_T}$ propagation steps.

Lastly, we can consider the special case of unitary dynamics. No storage at all is required in the semi-AD scheme: after the initial forward propagation in Fig. 1 (a), the resulting $|\Psi_k(T)\rangle$ can simply be backward-propagated in parallel with $|\tilde{\chi}_k(T)\rangle$. Thus, we trade the need for storage with a full additional time propagation. The method is not applicable in open quantum systems where the dynamics are not unitary, and thus a backward propagation of $|\Psi_k(T)\rangle$ does not recover the states $|\Psi_k(t_n)\rangle$ from the forward propagation. The unitary dynamics can be exploited in the same way to reduce the storage in a full-AD implementation of GRAPE [101].

The asymptotic memory requirement for both full-AD and semi-AD for different propagators and with and without checkpointing is summarized in Table 1. It is important to point out that for semi-

	Full-AD	Full-AD	Full-AD	Full-AD	Semi-AD	Semi-AD
prop. meth.	exp.	exp.	polyn.	polyn.	any	any
checkpoint		✓		✓		✓
matrix (N_H^2)	N_T	$N_C + C$	MN_T	$N_C + MC$	0	0
vector (N_H)	N_T	$N_C + C$	MN_T	$N_C + MC$	N_T	$N_C + C$
mem ($\mathcal{O}$)	$N_T N_H^2$	$(N_C + C)N_H^2$	$MN_T N_H^2$	$(N_C + MC)N_H^2$	$N_T N_H$	$(N_C + C)N_H$

Table 1: Asymptotic memory usage for full automatic differentiation (Full-AD) and semi-automatic differentiation (Semi-AD). The table shows how the number of matrices and vectors that must be stored in memory scales with the size of the Hilbert space N_H and the number of time steps N_T. For full-AD, we compare propagation via matrix exponentiation ("exp.") and propagation via a polynomial method ("polyn."), such as the Chebychev polynomials detailed in section 2.3. In this case M is the order of the polynomial required for convergence, equal to the number of matrix-vector products in a propagation step. Also included is the number of matrices/states that need to be stored when checkpointing is used. C denotes the time steps between checkpoints, and $N_C = N_T/C$ is the number of checkpoints. The bottom row shows the total asymptotic scaling of the required memory.

AD, the memory requirement is determined only from the storage of states, and is thus linear both in the Hilbert space dimension and in the number of time steps. The remaining use of automatic differentiation to evaluate $\nabla_{\Psi_k} J_T$ in Eq. (31), $\nabla_{\tau_k} J_T$ in Eq. (32), or $\nabla_{U_L} J_T$ in Eq. (37) is negligible in comparison. At worst, for ∇_{Ψ_k}, it is a constant number of states. At best, for ∇_{τ_k}, it is a small number of scalars.

In practice, the memory and runtime characteristics of full-AD may be less predictable. Many AD frameworks have large constant overhead and large prefactors for the asymptotic scaling in Table 1. On the other hand, the size of the computational graph and thus the amount of memory required for AD depend heavily on the exact implementation details of the time propagation. Therefore, we will explore the scaling of memory and runtime empirically for a specific implementation and a realistic example in Section 5.

As discussed in Section 3, there is a certain freedom to define what operations constitute "elementary operation", with a known pre-defined adjoint. For example, at least in principle it would be possible to define an entire propagation step via Chebychev expansion as a single node in the computational graph. This would eliminate the scaling with the number of coefficients M, but would require to implement a custom adjoint for that propagation step, which is not trivial. Custom adjoints would have to be implemented by hand for every different propagation method.

In general, any use of automatic differentiation involves a trade-off between memory usage, runtime, and code complexity (with custom adjoints). We believe that semi-automatic differentiation is a particularly attractive balance of these three goals: It is strictly linear in the number of time steps and the dimension of the Hilbert space, its runtime and structure match that of a traditional GRAPE implementation without any AD capabilities, and it requires no implementation of any custom adjoints, which may otherwise achieve similar memory scaling. Specifically, it works with any propagation method without modification; the runtime of the optimization is directly proportional to the runtime of the time propagation.

5 Optimizing Gate Concurrence for Two Coupled Transmons

5.1 Two-Qubit Gates on Transmons with a Shared Transmission Line

As an example to benchmark the semi-automatic differentiation approach to optimal control we consider two superconducting transmon qubits [72] with a shared transmission line [73] ("cavity") that allows to control the system via microwave pulses. Each transmon qubit is an anharmonic Duffing

left qubit frequency	ω_1	=	4.380	·	2π GHz
right qubit frequency	ω_2	=	4.614	·	2π GHz
rotating frame (drive) frequency	ω_d	=	4.498	·	2π GHz
left qubit anharmonicity	α_1	=	210	·	2π MHz
right qubit anharmonicity	α_1	=	215	·	2π MHz
effective qubit-qubit coupling	J	=	-3	·	2π MHz
relative coupling strength	λ	=	1.03		

Table 2: Parameters for the system of two coupled transmon qubits.

oscillator that couples to the transmission line. In the dispersive limit, where the qubit-cavity detuning dominates the coupling strength, the cavity can be eliminated. This results in an effective two-transmon Hamiltonian with a static qubit-qubit coupling. Furthermore, a microwave control field with a frequency ω_d near the qubit frequencies ω_1, ω_2 drives transitions on the transmons. In the rotating-wave approximation, the effective Hamiltonian reads [102]

$$\hat{\mathsf{H}} = \hat{\mathsf{H}}_0 + \Omega_{\mathrm{re}}(t)\hat{\mathsf{H}}_{d,\mathrm{re}} + \Omega_{\mathrm{im}}(t)\hat{\mathsf{H}}_{d,\mathrm{im}} \tag{49a}$$

with ($\hbar = 1$)

$$\hat{\mathsf{H}}_0 = \sum_{q=1,2}\left[\left(\omega_q - \omega_d + \frac{\alpha_q}{2}\right)\hat{\mathsf{b}}_q^\dagger\hat{\mathsf{b}}_q - \frac{\alpha_q}{2}(\hat{\mathsf{b}}_q^\dagger\hat{\mathsf{b}}_q)^2\right] + J\left(\hat{\mathsf{b}}_1^\dagger\hat{\mathsf{b}}_2 + \hat{\mathsf{b}}_1\hat{\mathsf{b}}_2^\dagger\right) \tag{49b}$$

$$\hat{\mathsf{H}}_{d,\mathrm{re}} = \frac{1}{2}\left[(\hat{\mathsf{b}}_1^\dagger + \hat{\mathsf{b}}_1) + \lambda(\hat{\mathsf{b}}_2^\dagger + \hat{\mathsf{b}}_2)\right] \tag{49c}$$

$$\hat{\mathsf{H}}_{d,\mathrm{im}} = \frac{\mathrm{i}}{2}\left[(\hat{\mathsf{b}}_1^\dagger - \hat{\mathsf{b}}_1) + \lambda(\hat{\mathsf{b}}_2^\dagger - \hat{\mathsf{b}}_2)\right] \tag{49d}$$

where $\hat{\mathsf{b}}_q^\dagger$ and $\hat{\mathsf{b}}_q$ are the creation and annihilation operators for the transmon excitations. Transmon qubits can be engineered to a wide range of frequencies, anharmonicities, and coupling strengths [103]. Here, we use the parameters listed in Table 2, cf. Ref. [35], as a typical example.

The control field in general is complex-valued, corresponding to variations in both the amplitude and phase relative to the rotating frame; it is easiest to split it into real and imaginary parts and treat them as independent controls $\Omega_{\mathrm{re}}(t)$ and $\Omega_{\mathrm{im}}(t)$, as we have done above.

The Hamiltonian in Eq. (49) is well-suited as a system on which to benchmark optimal control methods for quantum gates. First, the system is fully controllable, allowing to implement any two-qubit gate if no further restrictions are placed on the control field [34, 104]. Second, entangling quantum gates have been demonstrated with gate durations anywhere between 20 ns and 5 µs [35, 103, 105, 106]. Thus, we can reasonably explore the numerical scaling of the optimization procedure with the number of time steps. Lastly, the number of levels in the anharmonic oscillator that reasonably contribute to the gate dynamics varies significantly depending on the gate mechanism and the amplitudes and frequencies of the control field [106]. This allows us to benchmark the optimization for varying Hilbert space sizes. Especially for non-analytic controls obtained with optimal control, leakage from the logical subspace can be a significant problem, and we include as many as $N_q = 15$ levels in the numerical simulation to account for this.

5.2 Optimization Functionals

A primary benefit of the semi-automatic differentiation is that it allows optimizing arbitrary figures of merit, including ones for which it is difficult or impossible to derive analytical gradients. This freedom can significantly enhance the effectiveness of optimal control. For example, in the context of entangling quantum gates, it was demonstrated that optimizing for an arbitrary perfectly entangling quantum gate is easier than optimizing for a *specific* entangling gate such as CNOT [34, 35]. This is because for

a given Hamiltonian it is not known a priori *which* perfect entangler will be easiest to implement with given resources such as a maximum power of the control pulse. At the same time, for the realization of a universal quantum computer, one perfect entangler together with arbitrary single-qubit operations is sufficient.

The entangling power of a quantum gate is measured by the gate concurrence, defined as the maximum concurrence that can be obtained by applying the quantum gate to a separable input state [31]. It can be evaluated by writing a two-qubit gate $\hat{\mathsf{U}}_\mathrm{L}$ (a 4×4 matrix in the logical subspace) in a Cartan decomposition [32],

$$\hat{\mathsf{U}}_\mathrm{L} = \hat{\mathsf{k}}_1 \exp\left[\frac{\mathrm{i}}{2}(c_1\,\hat{\sigma}_x\hat{\sigma}_x + c_2\,\hat{\sigma}_y\hat{\sigma}_y + c_3\,\hat{\sigma}_z\hat{\sigma}_z)\right]\hat{\mathsf{k}}_2\,, \tag{50}$$

where $\hat{\sigma}_{x,y,z}$ are the Pauli matrices, $\hat{\mathsf{k}}_{1,2}$ are single-qubit operations, and $c_{1,2,3}$ are real-valued coefficients that characterize the two-qubit aspect of the quantum gate. When eliminating symmetries in Eq. (50), the coefficients can be understood as coordinates in a geometric representation of two-qubit gates called the Weyl chamber. A gate is a perfect entangler with a gate concurrence $C = 1$ if $c_1 + c_2 \geq \frac{\pi}{2}$, $c_1 - c_2 \leq \frac{\pi}{2}$, and $c_2 + c_3 \leq \frac{\pi}{2}$, which is a polyhedron within the Weyl chamber. Otherwise,

$$C(\hat{\mathsf{U}}_\mathrm{L}) = \max\left|\sin(c_{1,2,3} \pm c_{3,1,2})\right|\,, \tag{51}$$

cf. Ref. [31]. The calculation of the Weyl chamber coordinates themselves is described in Ref. [107] and implemented in Refs. [108, 109] and involves obtaining the eigenvalues of $\hat{\mathsf{U}}_\mathrm{L}$, as well as a branch selection of a complex logarithm. Thus, the evaluation of Eq. (51) is inherently non-analytical, preventing the analytic construction of a gradient ∇C with respect to the control values.

For this reason, a less direct measure for the entangling power of the quantum gate was developed in Refs. [34, 35]. Intuitively, it minimizes the geometric distance from the polyhedron of perfect entanglers in the Weyl chamber. Mathematically, that distance is formulated not in terms of the Weyl chamber coordinates, but in terms of the Makhlin local invariants $g_{1,2,3}$ [110]. Similarly to the Weyl chamber coordinates, these characterize two-qubit gates up to single-qubit operations. It can be shown that the geometric distance to the polyhedron of perfect entanglers is

$$D_\mathrm{PE}(\hat{\mathsf{U}}_\mathrm{L}) = g_3\sqrt{g_1^2 + g_2^2} - g_1\,. \tag{52}$$

Unlike the Weyl chamber coordinates, the local invariants can be calculated *analytically* from the two-qubit gate $\hat{\mathsf{U}}_\mathrm{L}$ as

$$g_1 = \frac{1}{16}\mathrm{Re}\left[\mathrm{tr}^2(\hat{\mathsf{m}})\right]\,, \qquad g_2 = \frac{1}{16}\mathrm{Im}\left[\mathrm{tr}^2(\hat{\mathsf{m}})\right]\,, \qquad g_3 = \frac{1}{4}\left[\mathrm{tr}^2(\hat{\mathsf{m}}) - \mathrm{tr}(\hat{\mathsf{m}}^2)\right]\,, \tag{53}$$

with $\hat{\mathsf{m}} = \hat{\mathsf{U}}_\mathrm{B}^T\hat{\mathsf{U}}_\mathrm{B}$, where $\hat{\mathsf{U}}_\mathrm{B}$ is the representation of $\hat{\mathsf{U}}_\mathrm{L}$ in the Bell basis. Applying matrix calculus, it is possible – although both tedious and lengthy – to calculate an analytic gradient of Eq. (52) [91]. For the derivative with respect to a state, cf. Eqs. (14, 29), a Python implementation is available in Ref. [108].

Both $C(\hat{\mathsf{U}}_\mathrm{L})$ and $D_\mathrm{PE}(\hat{\mathsf{U}}_\mathrm{L})$ are only well-defined if $\hat{\mathsf{U}}_\mathrm{L}$ is unitary. To ensure this in the optimization, we may add a term that calculates the loss of population from the logical subspace,

$$p_\mathrm{loss}(\hat{\mathsf{U}}_\mathrm{L}) = 1 - \frac{1}{4}\mathrm{tr}\left[\hat{\mathsf{U}}_\mathrm{L}^\dagger\hat{\mathsf{U}}_\mathrm{L}\right]\,. \tag{54}$$

Altogether, we use the following three optimization functionals:

1. Square-modulus gate optimization (SM)

$$J_{T,\mathrm{sm}}(\{\tau_k\}) = 1 - \left|\frac{1}{4}\sum_{k=1}^{4}\underbrace{\langle\phi_k^\mathrm{tgt}\,|\,\Psi(T)\rangle}_{\equiv\tau_k}\right|^2 = 1 - \frac{1}{16}\sum_{k=1}^{4}\sum_{k'=1}^{4}\tau_{k'}^*\tau_k\,, \tag{55}$$

cf. Eq. (2), where $|\phi_k^{\rm tgt}\rangle$ is the result of applying the target gate

$$\hat{\mathrm{O}} = \sqrt{\mathrm{iSWAP}} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{\mathrm{i}}{\sqrt{2}} & 0 \\ 0 & \frac{\mathrm{i}}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{56}$$

to the initial states $|\phi_1\rangle = |00\rangle$, $|\phi_2\rangle = |01\rangle$, $|\phi_3\rangle = |10\rangle$, $|\phi_4\rangle = |11\rangle$ spanning the logical subspace. The choice of $\sqrt{\mathrm{iSWAP}}$ as the target gate is somewhat arbitrary, although it has been demonstrated that this is an easily reachable gate for the Hamiltonian in Eq. (49) [35].

2. Perfect entangler optimization (PE)

$$J_{T,\mathrm{PE}}(\hat{\mathrm{U}}_\mathrm{L}) = \frac{1}{2}\left(1 - D_\mathrm{PE}(\hat{\mathrm{U}}_\mathrm{L})\right) + \frac{1}{2} p_\mathrm{loss}(\hat{\mathrm{U}}_\mathrm{L})\,, \tag{57}$$

where D_PE and p_loss are defined in Eq. (52) and Eq. (54), respectively.

3. Concurrence optimization (C)

$$J_{T,\mathrm{C}}(\hat{\mathrm{U}}_\mathrm{L}) = \frac{1}{2}\left(1 - C(\hat{\mathrm{U}}_\mathrm{L})\right) + \frac{1}{2} p_\mathrm{loss}(\hat{\mathrm{U}}_\mathrm{L})\,, \tag{58}$$

where C and p_loss are defined in Eq. (51) and Eq. (54), respectively. This is an example of a non-analytic functional whose gradient can only be evaluated via automatic differentiation.

5.3 Benchmarks

The result of benchmarking the functionals in Section 5.2 with semi-automatic and full automatic differentiation is shown in Fig. 2. The "Semi-AD" optimization uses the approach described in Section 4.2 for the gate optimization with the square-modulus functional (SM), and the approach described in Section 4.3 for the perfect-entanglers (PE) and concurrence (C) optimizations, as indicated by the argument on the left-hand-sides of Eqs. (55–58). The resulting optimized control fields are similar to those obtained in Ref. [35], and are shown in the "PE" examples in the online documentation of the `GRAPE.jl` and `Krotov.jl` packages [69, 70], which implement both the direct and semi-AD optimization methods.

All of the optimizations use an expansion in Chebychev polynomials for the time propagation, Eq. (7). We compare this with an optimization using full automatic differentiation ("Full-AD"). This means that we simply propagate the set of initial states with an AD-aware ODE solver, the Runge-Kutta algorithm DP5 [111] in `DifferentialEquations.jl` [112] and then evaluate the functional within the Zygote AD framework [61]. As an alternative to the general ODE solver, we also benchmark a full-AD variant of the Chebychev propagator. In this implementation, in-place linear algebra operations must be avoided, adding runtime overhead due to the allocation and deallocation of memory, cf. panels (e, f). It is possible to do this within Zygote without too much numerical overhead due to the simplicity of Eqs. (7, 8). Of course, the propagator is also limited to standard Hermitian dynamics. Lastly, as a baseline, we benchmark the direct optimization of the square-modulus functional for a $\sqrt{\mathrm{iSWAP}}$ gate with the analytic gradient in Eq. (11), that is, without any use of automatic differentiation ("Direct (Cheby) (SM)").

In the left column, panels (a, c, e), we vary the number of levels at which we cut off the transmon qubit between $N_q = 3$ and $N_q = 15$ levels, which we show in terms of the Hilbert space size, $N_H = N_q^2$ for two transmons. The gate duration is constant at 100 ns. In the right column, panels (b, d, f), we vary the gate duration between 20 ns and 800 ns while keeping the number of transmon levels constant at 5 ($N_H = 25$). The step size of the time grid is 0.1 ns, so that the number of time steps in the scheme of Fig. 1 (a) is directly proportional to the gate duration and varies between 200 and 8000 steps.

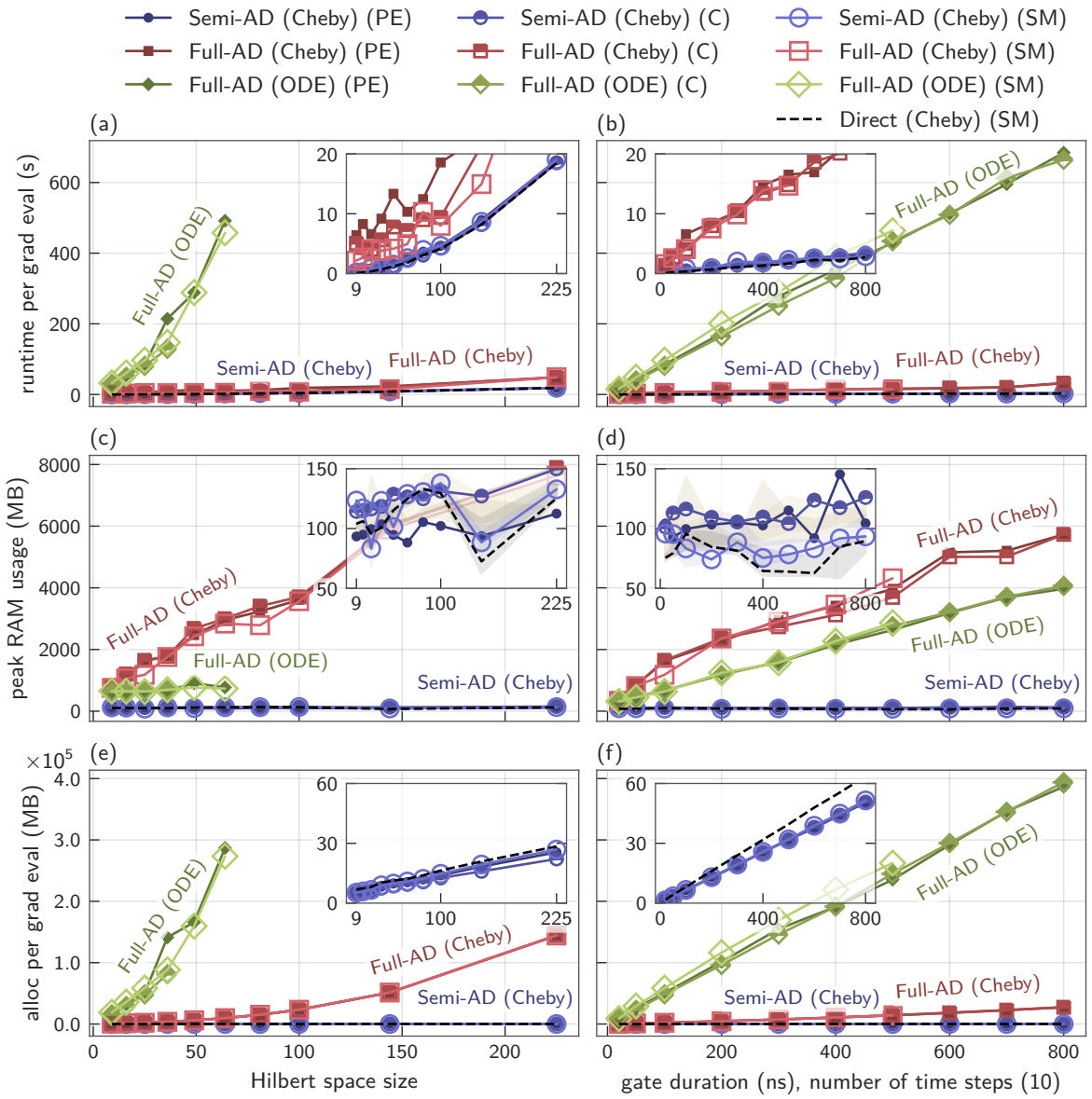

Figure 2: Benchmarks for 10 iterations of a GRAPE optimization of entangling gates on two coupled transmons for different functionals and different usage of automatic differentiation. The top row shows the optimization runtime in seconds per evaluation of the gradient (forward propagation of the basis states and backward propagation of the extended gradient-states); the center row shows the median peak RAM usage in megabyte from up to 20 optimization runs, where the insets also indicate the range of peak RAM usage as shaded areas; and the bottom row shows the memory allocations per gradient evaluation in megabyte. The left column shows how these quantities vary with the size of the Hilbert space; the right column shows the data for different gate durations in nanoseconds. This is directly proportional to the number of time steps ($dt = 0.1$ ns). Each panel shows the performance of an optimization towards a perfect entangler in the Weyl chamber (PE) or by maximizing the gate concurrence (C) directly. Additionally, the data for the optimization of a $\sqrt{\text{iSWAP}}$ gate via a square-modulus (SM) functional is shown. The dynamics are simulated by evaluating the time evolution operator in Chebychev polynomials ("Cheby"), or by using a generic ODE solver. The gradient is evaluated either via semi-automatic differentiation ("Semi-AD") for the PE and C optimizations or via a set over overlaps $\{\tau_k\}$ for the SM optimization, via full automatic differentiation ("Full-AD"), or – for the "Direct" optimization – using an analytic gradient (no automatic differentiation).

For each optimization, we run the optimization for 10 iterations. At a minimum, each iteration includes one evaluation of the gradient as depicted in Fig. 1 (a), that is, a forward propagation of the four logical basis states and a backward propagation of four extended gradient states. However, the linesearch may require additional evaluations of the gradient in order to determine the step width. The number of linesearch steps varies for different optimizations and between iterations. Thus, the benchmark of the runtime in panels (a,b) shows the seconds required per gradient evaluation. The optimizations were performed on an Intel Xeon Gold 6226R CPU workstation with a nominal clock speed of 2.9 GHz, without parallelization. The runtime was determined using `BenchmarkTools.jl` [113] as the minimum of 20 optimization runs, with a maximum total benchmark time of 24 hours. It excludes compilation overhead.

Fundamentally, the runtime is super-linear with the size of the Hilbert space. This is due to two factors: First, an increase in the spectral range, which for the Chebychev propagation implies an increase in the number of coefficients required for Eq. (7) to converge to machine precision. This increase is roughly linear with the number of levels in the transmon, i.e., the square root of the Hilbert space dimension. Second, the numerical scaling of the matrix-vector multiplications in Eq. (8) (and likewise in the implementation of a Runge-Kutta ODE solver). In principle, matrix-vector multiplication scales as N_H^2, although this is mitigated by the fact that we use sparse matrices to store the Hamiltonian. For the gate duration, the runtime is directly proportional to the number of time steps.

We find that the runtime of the Semi-AD optimization is virtually indistinguishable from that of the direct optimization. This is in stark contrast to the full-AD optimization with a general ODE solver. For a Hilbert space dimension greater than 64, the runtime of this optimization becomes prohibitively expensive (greater than 10 minutes per iteration, or step in the line search). The situation improves dramatically when using a modified Chebychev propagator for the full-AD optimization. Within the shown parameter region, these optimizations are well within the order of one minute per iteration. However, the insets in panels (a, b) show a significant difference in scaling between the semi-AD and full-AD Chebychev optimizations. For the linear scaling in panel (b), we find a factor of 10 between the different slopes (3 ms per time step versus 30 ms per time step). This order of magnitude also appears realistic for the scaling with respect to the Hilbert space dimension, and may become a problem for the runtime of much larger systems. Certainly, it will not be possible to perform a full-AD optimization of anything but the most trivial open quantum systems, with a Liouville space that is quadratically larger than the underlying Hilbert space. This is especially true because the Chebychev propagator is not applicable to non-unitary (open system) dynamics. The use of Newton polynomials discussed in Section 2.3 is numerically more demanding than the use of Chebychev polynomials, and it is unclear whether it would be feasible to implement a variant of the method that avoids in-place operations and would be compatible with a framework for automatic differentiation such as Zygote. Thus, a full-AD optimization of an open quantum system would likely have to rely on an ODE solver, with the prohibitive runtime shown in panel (a). The relative scaling of a direct optimization compared to a full-AD optimization matches recent observations in Ref. [114].

There is no substantial difference in runtime between the different optimization functionals. This illustrates that the numerical effort of the optimization is entirely dominated by the time propagation, and further explains why the performance of the Semi-AD optimization is indistinguishable from a direct optimization with an analytic gradient.

The peak RAM usage shown in panels (c, d) was measured by monitoring the Julia process running the optimization with `psutil` [115], and subtracting a baseline from a "Hello World" program to account for the footprint of the Julia runtime, which can vary depending on the exact version of Julia and the installed packages. We show the median RAM usage, as well as the range of values in the inset for the Semi-AD (Cheby) optimization. There is significant fluctuation in the peak RAM usage, as it depends on Julia's garbage collector, which has some element of randomness. However, the data shown in the insets indicates that the peak RAM usage for semi-automatic differentiation is essentially constant around 100 MB, and equal to the memory used for the direct optimization with an analytic gradient. In principle, as analyzed in Section 4.6, we would expect a linear scaling for large Hilbert space dimensions,. This is determined by the storage of the forward-propagated states marked in red

in Fig. 1 (a). For $N = 4$ targets, a Hilbert space dimension of N_H, and N_T time steps, each stored state is a vector of complex numbers of length N_H, and each complex number requires 64 bits for both the real and imaginary part, i.e., 16 bytes in total. Thus, the total memory required for storage is

$$\mathrm{MB}_{\mathrm{storage}} = N \cdot N_H \cdot N_T \cdot \frac{16}{(1024)^2} \,. \tag{59}$$

For $N_H = 225$, $N_T = 1000$, the rightmost point in the inset of panel (a), this comes out to 14 MB, and 49 MB for $N_H = 100$, $N_T = 8000$, the rightmost point in the inset of panel (b), and we can conclude that for the data shown in Fig. 2, the propagation overhead still dominates over the storage of propagated states.

The memory usage a full-AD optimization with an ODE solver appears constant with respect to the size of the Hilbert space, albeit with a significant overhead, averaging around 700 MB. This is likely due to the solver being optimized for automatic differentiation, that is, providing handwritten adjoints for the Runge-Kutta step and thus eliminating the overhead of the computational graph for a single propagation step. Memory usage still scales linearly with the number of time steps, reaching 4 GB for 8000 time steps. The full-AD optimization using a Chebychev expansion does not benefit from the AD-aware propagation in `DifferentialEquations.jl`, and thus the memory usage scales with the number of coefficients in Eq. (7), resulting in an excessive RAM usage of 8 GB for a Hilbert space dimension of 225. Thus, the use of the Chebychev propagator for significantly larger Hilbert spaces would be prohibitive, and a possible implementation of a Newton propagator would likely perform even worse, making the use of full-AD for open quantum systems impractical.

Lastly, in panels (e, f) we show the accumulated memory allocated on the heap, as measured by `BenchmarkTools.jl`. Note the scale of the y-axis, which reaches 4×10^5 MB, i.e., 400 GB. This is normalized by the number of gradient evaluations. The allocations differ from the peak RAM usage due to Julia's garbage collector, but correlate strongly with the runtime shown in panels (a, b). This illustrates the importance of good memory management, which is easy for an in-place Chebychev propagator (the negligible allocations of < 60 MB shown in the inset), but impossible for a propagator running in an automatic differentiation framework that does not allow for in-place operations.

6 Conclusion and Outlook

We have developed a theory of semi-automatic differentiation that allows to optimize arbitrary functionals in an efficient and flexible manner. Separating time propagation and the evaluation of the functional eliminates the excessive computational overhead and limited scope traditionally associated with the use of automatic differentiation (AD). With semi-AD, the time propagation and the associated gradient can be evaluated outside of the AD framework in a modified GRAPE scheme, which we have described in detail. This scheme can be implemented in the most efficient manner possible, using sparse linear algebra, complex matrices, and in-place operations. The remaining part of the gradient is of minimal computational complexity and can thus be efficiently evaluated within an AD framework.

For the example of entangling gates on superconducting transmon qubits, we have verified that we can optimize for an arbitrary perfectly entangling quantum gate, either via a functional exploiting the geometric structure of the Weyl chamber, or by directly by maximizing the gate concurrence. This is the first demonstration of a gradient-based optimization of the concurrence, or any non-analytic functional. Fundamentally, such functionals *require* the use of automatic differentiation and have been held back thus far by the associated numerical overhead.

The runtime and memory usage of the semi-AD optimizations shown here scales identically to a direct optimization of a quantum gate with a fully analytic gradient. That is, we completely eliminate the exorbitant numerical overhead traditionally associated with automatic differentiation. A "full-AD" optimization that uses a generic ODE solver within the AD framework becomes unfeasible in terms of runtime for Hilbert space dimensions greater than ≈ 100. The runtime can be improved significantly by adapting a propagation via expansion into Chebychev polynomials to the requirements of the AD

framework (no in-place linear algebra operations). However, this results in excessive memory usage and would be difficult to extend to the significantly more complicated propagators required for open quantum systems.

As we have demonstrated, the potential improvements of semi-automatic differentiation scale super-linearly both for runtime and memory usage, compared to a full-AD approach. Even for moderate Hilbert space sizes in a closed quantum system, we observe improvements of up to two orders of magnitude. For control problems of larger dimension (either the dimension of the Hilbert space or the number of control parameters), and especially in open quantum systems, that effect will be further magnified. In such a setting, and for functionals where an analytic gradient is not feasible, semi-automatic differentiation will be the only viable option. Fundamentally, an optimization will be possible as long as simulating the time dynamics of the system is computationally feasible.

We have implemented the semi-AD approach in two ready-to-use Julia packages, `GRAPE.jl` [69] and `Krotov.jl` [70], as part of the more general `QuantumControl.jl` [71]. As an AD framework, we have used Zygote [60, 61]. However, the method described here is applicable to any language or AD framework. In fact, the low complexity of the evaluation of the optimization functional relative to the full time propagation allows for additional avenues in environments where AD is underdeveloped. For example, we have tested the use of finite differences [116] and found it to be adequate for the examples in Section 5. Similarly, we would expect the easier-to-implement forward-mode differentiation to be a practical alternative.

We have developed the method of semi-automatic differentiation for the standard GRAPE model of piecewise-constant control fields, albeit extending it to arbitrary functionals. However, the idea applies also to extensions of GRAPE for a reduced number of control parameters, as used in the GOAT [117], GROUP [118], and GRAFS [119] methods. Parametrization of the control field may be taken into account by a further chain rule in Eq. (3). Furthermore, as pointed out in Ref. [117], Eq. (5) can be used not just to evaluate the gradient of a piecewise constant time step, but also the gradient a full time propagation with respect to a single control parameter. In all of these cases, the method can be augmented with semi-automatic differentiation to extend it to arbitrary functionals. We will explore this in future work as part of the `QuantumControl.jl` framework [71].

In addition to enabling the use of AD, a second motivation for using a framework like Tensorflow to model the entire optimization problem was to enable the use of GPU computing, enabling considerable speedups [37, 49]. This possibility remains with semi-automatic differentiation, but the use of AD and GPU computing are now entirely independent: The time propagation can be implemented in whatever way is most efficient, including on the GPU.

Lastly, in Ref. [120], it was observed that the scheme in Fig. 1 (a) can be extended to compute the full Hessian of an overlap of two states. With next-generation AD systems that allow for the calculation of higher-order derivatives [121], this opens up the possibility of a full Hessian semi-automatic differentiation approach. The resulting Newton optimization may provide better convergence than the pseudo-Newton achievable via LBFGS that we have used here.

Application of the semi-automatic differentiation framework developed in this paper will open new pathways to solve long-standing problems in quantum control. Apart from the optimization of entanglement measures in quantum information, which we have demonstrated here, and which could be extended to open quantum systems [122], the method would be applicable to the creation of multipartite entanglement in many-body systems. To date, this has been addressed with optimal control, either indirectly [123] or with gradient-free methods [124], usually via the CRAB method [125–127]. The ability to explore the optimization landscape for these control problems more broadly may lead to deep insights into the quantum behavior, e.g., of biological systems [128, 129].

In quantum metrology, there is a wide range of opportunities to use optimal control beyond simple state-to-state transfers. For one, we can address functionals like the recently developed population transfer functional for atom interferometry [130]. So far, this has only been explored with Krotov's method, but the use of semi-AD would allow optimizing the components of an atom interferometer using GRAPE, potentially exploiting the improved asymptotic convergence [24] in the generally flat optimization landscape of a robustness optimization [89]. This would also extend to a recently proposed

tractor atom interferometer [131, 132]. More fundamentally, we may directly maximize metrological measures through quantum control [133, 134]. Generalizing from a recent application of optimal control to the creation of extreme spin-squeezed states [135] we may wish to directly maximize the quantum Fisher information [28, 29], which is non-analytic in open quantum systems [30]. Thus, we expect semi-automatic differentiation to be an indispensable tool for the design of metrological protocols in open quantum systems.

Data Availability

The code used to generate the benchmarks in Section 5 and Fig. 2 is available at Ref. [136], or under DOI 10.5281/zenodo.7386493. Moreover, examples for the perfect entangler and concurrence optimizations for coupled transmon qubits that show the resulting optimized fields are available as part of the documentation of the `GRAPE.jl` and `Krotov.jl` packages, which implement the semi-AD approach [69, 70].

Acknowledgments

MHG and SCC acknowledge support by the DEVCOM Army Research Laboratory under Cooperative Agreement Number W911NF-16-2-0147 and W911NF-21-2-0037, respectively. The work was also supported by DEVCOM Army Research Laboratory through DIRA-TRC No. DTR19-CI-019. The authors thank Alastair Marshall for contributions to the `GRAPE.jl` package.

References

[1] Constantin Brif, Raj Chakrabarti, and Herschel Rabitz. "Control of quantum phenomena: past, present and future". New J. Phys. **12**, 075008 (2010).
[2] Moshe Shapiro and Paul Brumer. "Quantum control of molecular processes". Wiley & Sons. (2012). Second edition.
[3] Christiane P. Koch. "Controlling open quantum systems: tools, achievements, and limitations". J. Phys.: Condens. Matter **28**, 213001 (2016).
[4] Ignacio R. Sola, Bo Y. Chang, Svetlana A. Malinovskaya, and Vladimir S. Malinovsky. "Quantum control in multilevel systems". In Ennio Arimondo, Louis F. DiMauro, and Susanne F. Yelin, editors, Advances In Atomic, Molecular, and Optical Physics. Volume 67, chapter 3, pages 151–256. Academic Press (2018).
[5] Oleg V. Morzhin and Alexander N. Pechen. "Krotov method for optimal control of closed quantum systems". Russ. Math. Surv. **74**, 851 (2019).
[6] Frank K. Wilhelm, Susanna Kirchhoff, Shai Machnes, Nicolas Wittler, and Dominique Sugny. "An introduction into optimal control for quantum technologies" (2020). arXiv:2003.10132.
[7] Christiane P. Koch, Ugo Boscain, Tommaso Calarco, Gunther Dirr, Stefan Filipp, Steffen J. Glaser, Ronnie Kosloff, Simone Montangero, Thomas Schulte-Herbrüggen, Dominique Sugny, and Frank K. Wilhelm. "Quantum optimal control in quantum technologies. strategic report on current status, visions and goals for research in Europe". EPJ Quantum Technol. **9**, 19 (2022).
[8] Michael Nielsen and Isaac L. Chuang. "Quantum computation and quantum information". Cambridge University Press. (2000).
[9] Iulia M. Georgescu, Sahel Ashhab, and Franco Nori. "Quantum simulation". Rev. Mod. Phys. **86**, 153 (2014).
[10] Christian L. Degen, Friedemann Reinhard, and Paola Cappellaro. "Quantum sensing". Rev. Mod. Phys. **89**, 035002 (2017).
[11] Ugo Boscain, Mario Sigalotti, and Dominique Sugny. "Introduction to the Pontryagin maximum principle for quantum optimal control". PRX Quantum **2**, 030203 (2021).

[12] Navin Khaneja, Timo Reiss, Cindie Kehlet, Thomas Schulte-Herbrüggen, and Steffen J. Glaser. "Optimal control of coupled spin dynamics: design of NMR pulse sequences by gradient ascent algorithms". J. Magnet. Res. **172**, 296 (2005).
[13] Pierre de Fouquières, Sophie G. Schirmer, Steffen J. Glaser, and Ilya Kuprov. "Second order gradient ascent pulse engineering". J. Magnet. Res. **212**, 412 (2011).
[14] Vadim F. Krotov and N. N. Fel'dman. "An iterative method for solving optimal-control problems". Engrg. Cybernetics **21**, 123 (1983).
[15] Vadim F. Krotov. "A technique of global bounds in optimal control theory". Control Cybern. **17**, 115 (1988).
[16] Vadim F. Krotov. "Global methods in optimal control". Dekker. New York, NY, USA (1996).
[17] József Somlói, Vladimir A. Kazakov, and David J. Tannor. "Controlled dissociation of I_2 via optical transitions between the X and B electronic states". Chem. Phys. **172**, 85 (1993).
[18] Allon Bartana, Ronnie Kosloff, and David J. Tannor. "Laser cooling of internal degrees of freedom. II". J. Chem. Phys. **106**, 1435 (1997).
[19] José P. Palao and Ronnie Kosloff. "Optimal control theory for unitary transformations". Phys. Rev. A **68**, 062308 (2003).
[20] Daniel M. Reich, Mamadou Ndong, and Christiane P. Koch. "Monotonically convergent optimization in quantum control using Krotov's method". J. Chem. Phys. **136**, 104103 (2012).
[21] Amir Rashidinejad, Yihan Li, and Andrew M. Weiner. "Recent advances in programmable photonic-assisted ultrabroadband radio-frequency arbitrary waveform generation". IEEE J. Quantum Electron. **52**, 1 (2016).
[22] Andrew M. Weiner. "Femtosecond pulse shaping using spatial light modulators". Rev. Sci. Instr. **71**, 1929 (2000).
[23] J. Robert Johansson, Paul D. Nation, and Franco Nori. "QuTiP 2: A Python framework for the dynamics of open quantum systems". Comput. Phys. Commun. **184**, 1234 (2013).
[24] Shai Machnes, Uwe Sander, Steffen J. Glaser, Pierre de Fouquières, Audrūnas Gruslys, Sophie G. Schirmer, and Thomas Schulte-Herbrüggen. "Comparing, optimizing, and benchmarking quantum-control algorithms in a unifying programming framework". Phys. Rev. A **84**, 022305 (2011).
[25] Hannah J. Hogben, Matthew Krzystyniak, Gareth T. P. Charnock, Peter J. Hore, and Ilya Kuprov. "Spinach – a software library for simulation of spin dynamics in large spin systems". J. Magnet. Res. **208**, 179 (2011).
[26] Zdeněk Tošner, Thomas Vosegaard, Cindie Kehlet, Navin Khaneja, Steffen J. Glaser, and Niels Chr. Nielsen. "Optimal control in NMR spectroscopy: Numerical implementation in SIMPSON". J. Magnet. Res. **197**, 120 (2009).
[27] Michael H. Goerz, Daniel Basilewitsch, Fernando Gago-Encinas, Matthias G. Krauss, Karl P. Horn, Daniel M. Reich, and Christiane P. Koch. "Krotov: A Python implementation of Krotov's method for quantum optimal control". SciPost Phys. **7**, 080 (2019).
[28] Samuel L. Braunstein and Carlton M. Caves. "Statistical distance and the geometry of quantum states". Phys. Rev. Lett. **72**, 3439 (1994).
[29] Luca Pezzé and Augusto Smerzi. "Entanglement, nonlinear dynamics, and the Heisenberg limit". Phys. Rev. Lett. **102**, 100401 (2009).
[30] Jian Ma, Xiaoguang Wang, Changpu P. Sun, and Franco Nori. "Quantum spin squeezing". Phys. Rep. **509**, 89 (2011).
[31] Barbara Kraus and J. Ignacio Cirac. "Optimal creation of entanglement using a two-qubit gate". Phys. Rev. A **63**, 062309 (2001).
[32] Jun Zhang, Jiří Vala, Shankar Sastry, and K. Birgitta Whaley. "Geometric theory of nonlocal two-qubit operations". Phys. Rev. A **67**, 042313 (2003).
[33] Paul Watts, Maurice O'Connor, and Jiří Vala. "Metric structure of the space of two-qubit gates, perfect entanglers and quantum control". Entropy **15**, 1963 (2013).

[34] Paul Watts, Jiří Vala, Matthias M. Müller, Tommaso Calarco, K. Birgitta Whaley, Daniel M. Reich, Michael H. Goerz, and Christiane P. Koch. "Optimizing for an arbitrary perfect entangler: I. Functionals". Phys. Rev. A **91**, 062306 (2015).
[35] Michael H. Goerz, Giulia Gualdi, Daniel M. Reich, Christiane P. Koch, Felix Motzoi, K. Birgitta Whaley, Jiří Vala, Matthias M. Müller, Simone Montangero, and Tommaso Calarco. "Optimizing for an arbitrary perfect entangler. II. Application". Phys. Rev. A **91**, 062307 (2015).
[36] Hamza Jirari. "Optimal control approach to dynamical suppression of decoherence of a qubit". Europhys. Lett. **87**, 40003 (2009).
[37] Nelson Leung, Mohamed Abdelhafez, Jens Koch, and David I. Schuster. "Speedup for quantum optimal control from automatic differentiation based on graphics processing units". Phys. Rev. A **95**, 042318 (2017).
[38] Mohamed Abdelhafez, David I. Schuster, and Jens Koch. "Gradient-based optimal control of open quantum systems using quantum trajectories and automatic differentiation". Phys. Rev. A **99**, 052327 (2019).
[39] Mohamed Abdelhafez, Brian Baker, András Gyenis, Pranav Mundada, Andrew A. Houck, David I. Schuster, and Jens Koch. "Universal gates for protected superconducting qubits using optimal control". Phys. Rev. A **101**, 022321 (2020).
[40] Frank Schäfer, Michal Kloc, Christoph Bruder, and Niels Lörch. "A differentiable programming method for quantum control". Mach. Learn.: Sci. Technol. **1**, 035009 (2020).
[41] Nelson Leung and Contributors (2021). code: SchusterLab/quantum-optimal-control.
[42] Daniel Weiss and Contributors (2021). code: SchusterLab/qoc.
[43] Andreas Griewank and Andrea Walther. "Evaluating derivatives". Society for Industrial and Applied Mathematics. Philadelphia (2008). Second edition.
[44] Charles C. Margossian. "A review of automatic differentiation and its efficient implementation". WIREs Data Mining Knowl Discov.**9** (2019).
[45] Frank Schäfer, Pavel Sekatski, Martin Koppenhöfer, Christoph Bruder, and Michal Kloc. "Control of stochastic quantum dynamics by differentiable programming". Mach. Learn.: Sci. Technol. **2**, 035004 (2021).
[46] Thomas Propson, Brian E. Jackson, Jens Koch, Zachary Manchester, and David I. Schuster. "Robust quantum optimal control with trajectory optimization". Phys. Rev. Applied **17**, 014036 (2022).
[47] Michael H. Goerz and Kurt Jacobs. "Efficient optimization of state preparation in quantum networks using quantum trajectories". Quantum Sci. Technol. **3**, 045005 (2018).
[48] Nicolas Wittler, Federico Roy, Kevin Pack, Max Werninghaus, Anurag Saha Roy, Daniel J. Egger, Stefan Filipp, Frank K. Wilhelm, and Shai Machnes. "Integrated tool set for control, calibration, and characterization of quantum devices applied to superconducting qubits". Phys. Rev. Applied **15**, 034080 (2021).
[49] Harrison Ball, Michael J. Biercuk, Andre Carvalho, Jiayin Chen, Michael Hush, Leonardo A. De Castro, Li Li, Per J. Liebermann, Harry J. Slatyer, Claire Edmunds, Virginia Frey, Cornelius Hempel, and Alistair Milne. "Software tools for quantum control: Improving quantum computer performance through noise and error suppression" (2020). arXiv:2001.04060.
[50] Asad Raza and Contributors (2022). code: qgrad/qgrad.
[51] Andreas Griewank. "Who invented the reverse mode of differentiation". In Martin Grötschel, editor, Optimization Stories. Pages 389–400. Documenta Mathematica. 21st International Symposium on Mathematical Programming, Berlin (2012). url: www.zib.de/groetschel/publications/OptimizationStories.pdf.
[52] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". Nature **323**, 533 (1986).
[53] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. "Tensorflow: A system for

large-scale machine learning". In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). Page 265. (2016). url: www.tensorflow.org/.

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An imperative style, high-performance deep learning library". In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché Buc, Edward A. Fox, and Roman Garnett, editors, Advances in Neural Information Processing Systems 32. Pages 8024–8035. Vancouver, BC, Canada (2019). Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019. url: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[55] Adam Paszke and Contributors (2022). code: pytorch/pytorch.

[56] Roy Frostig, Matthew Johnson, and Chris Leary. "Compiling machine learning programs via high-level tracing". In SysML Conference. Stanford, CA (2018). url: mlsys.org/Conferences/2019/doc/2018/146.pdf.

[57] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang (2018). code: google/jax.

[58] Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. "Fashionable modelling with Flux" (2018). arXiv:1811.01457.

[59] Michael Innes. "Flux: Elegant machine learning with Julia". J. Open Source Softw. **3**, 602 (2018).

[60] Michael Innes. "Don't unroll adjoint: Differentiating SSA-form programs" (2018). arXiv:1810.07951.

[61] Michael Innes and Contributors (2022). code: FluxML/Zygote.jl.

[62] Hillel Tal-Ezer and Ronnie Kosloff. "An accurate and efficient scheme for propagating the time dependent Schrödinger equation". J. Chem. Phys. **81**, 3967 (1984).

[63] Ronnie Kosloff. "Time-dependent quantum-mechanical methods for molecular dynamics". J. Chem. Phys. **92**, 2087 (1988).

[64] Michael Berman, Ronnie Kosloff, and Hillel Tal-Ezer. "Solution of the time-dependent liouville-von neumann equation: dissipative evolution". J. Phys. A **25**, 1283 (1992).

[65] Ronnie Kosloff. "Propagation methods for quantum molecular dynamics". Annu. Rev. Phys. Chem. **45**, 145 (1994).

[66] Guy Ashkenazi, Ronnie Kosloff, Sanford Ruhman, and Hillel Tal-Ezer. "Newtonian propagation methods applied to the photodissociation dynamics of I_3^-". J. Chem. Phys. **103**, 10005–10014 (1995).

[67] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. "Julia: A fresh approach to numerical computing". SIAM Rev. **59**, 65 (2017).

[68] "The Julia programming language". url: julialang.org.

[69] Michael H. Goerz and Contributors (2022). code: JuliaQuantumControl/GRAPE.jl.

[70] Michael H. Goerz and Contributors (2022). code: JuliaQuantumControl/Krotov.jl.

[71] Michael H. Goerz and Contributors (2022). code: JuliaQuantumControl/QuantumControl.jl.

[72] Jens Koch, Terri M. Yu, Jay Gambetta, Andrew A. Houck, David I. Schuster, Johannes Majer, Alexandre Blais, Michel H. Devoret, Steven M. Girvin, and Robert J. Schoelkopf. "Charge-insensitive qubit design derived from the Cooper pair box". Phys. Rev. A **76**, 042319 (2007).

[73] Alexandre Blais, Jay Gambetta, A. Wallraff, D. I. Schuster, Steven M. Girvin, M. H. Devoret, and Robert J. Schoelkopf. "Quantum-information processing with circuit quantum electrodynamics". Phys. Rev. A **75**, 032329 (2007).

[74] Michael H. Goerz, Daniel M. Reich, and Christiane P. Koch. "Optimal control theory for a unitary operation under dissipative evolution" (2021). arXiv:1312.0111v2.

[75] The MathWorks. Natick, MA, USA. "Matlab optimization toolbox". (2018).

[76] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. "SciPy 1.0: Fundamental algorithms for scientific computing in Python". Nat. Methods **17**, 261 (2020).

[77] Eric Jones, Travis Oliphant, Pearu Peterson, et al. "SciPy: Open source scientific tools for Python". (2001–). url: docs.scipy.org/doc/scipy/.

[78] Patrick K. Mogensen and Asbjørn N. Riseth. "Optim: A mathematical optimization package for Julia". J. Open Source Softw. **3**, 615 (2018).

[79] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. "A limited memory algorithm for bound constrained optimization". SIAM J. Sci. Comput. **16**, 1190 (1995).

[80] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization". ACM Trans. Math. Softw. **23**, 550 (1997).

[81] Yupei Qi and Contributors (2022). code: Gnimuc/LBFGSB.jl.

[82] Patrick K. Mogensen, Asbjørn N. Riseth, and Contributors (2020). code: JuliaNLSolvers/LineSearches.jl.

[83] Charles F. Van Loan. "Computing integrals involving the matrix exponential". IEEE Trans. Automat. Contr. **23**, 395 (1978).

[84] David L. Goodwin and Ilya Kuprov. "Auxiliary matrix formalism for interaction representation transformations, optimal control, and spin relaxation theories". J. Chem. Phys. **143**, 084113 (2015).

[85] Amparo Gil, Javier Segura, and Nico M. Temme. "Numerical methods for special functions". Society for Industrial and Applied Mathematics. (2007).

[86] Hillel Tal-Ezer. "On restart and error estimation for Krylov approximation of $w = f(a)v$". SIAM J. Sci. Comput. **29**, 2426 (2007).

[87] Michael H. Goerz and Contributors (2022). code: JuliaQuantumControl/QuantumPropagators.jl.

[88] Michael H. Goerz, Daniel M. Reich, and Christiane P. Koch. "Optimal control theory for a unitary operation under dissipative evolution". New J. Phys. **16**, 055012 (2014).

[89] Michael H. Goerz, Eli J. Halperin, Jon M. Aytac, Christiane P. Koch, and K. Birgitta Whaley. "Robustness of high-fidelity Rydberg gates with single-site addressability". Phys. Rev. A **90**, 032329 (2014).

[90] Mamadou Ndong, Hillel Tal-Ezer, Ronnie Kosloff, and Christiane P. Koch. "A Chebychev propagator for inhomogeneous Schrödinger equations". J. Chem. Phys. **130**, 124108 (2009).

[91] Michael H. Goerz. "Optimizing robust quantum gates in open quantum systems". PhD thesis. Universität Kassel. (2015). url: d-nb.info/1072259729/34.

[92] Christopher Rackauckas. "Engineering trade-offs in automatic differentiation: from TensorFlow and PyTorch to Jax and Julia". url: http://www.stochasticlifestyle.com/engineering-trade-offs-in-automatic-differentiation-from-tensorflow-and-pytorch-to-jax-and-julia/.

[93] Jan R. Magnus and Heinz Neudecker. "Matrix differential calculus with applications in statistics and econometrics". Wiley Series in Probability and Statistics. Wiley. (2019). Third edition.

[94] Kaare B. Petersen and Michael S. Pedersen. "The matrix cookbook". Technical report. Technical University of Denmark (2012). url: http://www2.imm.dtu.dk/pubdb/p.php?3274.

[95] Mike B. Giles. "Collected matrix derivative results for forward and reverse mode algorithmic differentiation". In Advances in Automatic Differentiation. Volume 64, pages 35–44. Springer (2008).

[96] Mike B. Giles. "An extended collection of matrix derivative results for forward and reverse mode automatic differentiation". Technical Report NA-08-01. Oxford University Computing Laboratory (2008). url: people.maths.ox.ac.uk/gilesm/files/NA-08-01.pdf.

[97] Are Hjørungnes. "Complex-valued matrix derivatives: With applications in signal processing and communications". Cambridge University Press. (2011).
[98] L. Susan Blackford, James Demmel, Jack J. Dongarra, Ian S. Duff, Sven Hammarling, Greg Henry, Michael Heroux, Linda Kaufman, Andrew Lumsdain, Antoine Petitet, Roldan Pozo, Karin Remington, and R. Clint Whaley. "An updated set of basic linear algebra subprograms (BLAS)". ACM Trans. Math. Softw. **28**, 135 (2002).
[99] Sophie G. Schirmer and Pierre de Fouquières. "Efficient algorithms for optimal control of quantum dynamics: the Krotov method unencumbered". New J. Phys. **13**, 073029 (2011).
[100] José P. Palao, Ronnie Kosloff, and Christiane P. Koch. "Protecting coherence in optimal control theory: State-dependent constraint approach". Phys. Rev. A **77**, 063412 (2008).
[101] Sri H. K. Narayanan, Thomas Propson, Marcelo Bongarti, Jan Hueckelheim, and Paul Hovland. "Reducing memory requirements of quantum optimal control". Technical Report ANL/MCS-P9566-0222. Argonne National Laboratory (2022).
[102] Stefano Poletto, Jay M. Gambetta, Seth T. Merkel, John A. Smolin, Jerry M. Chow, A. D. Córcoles, George A. Keefe, Mary B. Rothwell, J. R. Rozen, D. W. Abraham, Chad Rigetti, and M. Steffen. "Entanglement of two superconducting qubits in a waveguide cavity via monochromatic two-photon excitation". Phys. Rev. Lett. **109**, 240505 (2012).
[103] Michael H. Goerz, Felix Motzoi, K. Birgitta Whaley, and Christiane P. Koch. "Charting the circuit QED design landscape using optimal control theory". npj Quantum Inf **3**, 37 (2017).
[104] Wen-Long Ma, Shruti Puri, Robert J. Schoelkopf, Michel H. Devoret, Steven M. Girvin, and Liang Jiang. "Quantum control of bosonic modes with superconducting circuits". Sci. Bull. **66**, 1789 (2021).
[105] Morten Kjaergaard, Mollie E. Schwartz, Jochen Braumüller, Philip Krantz, Joel I.-J. Wang, Simon Gustavsson, and William D. Oliver. "Superconducting qubits: Current state of play". Annu. Rev. Condens. Matter Phys. **11**, 369 (2020).
[106] Alexandre Blais, Arne L. Grimsmo, Steven M. Girvin, and Andreas Wallraff. "Circuit quantum electrodynamics". Rev. Mod. Phys. **93**, 025005 (2021).
[107] Andrew Childs, Henry Haselgrove, and Michael Nielsen. "Lower bounds on the complexity of simulating quantum gates". Phys. Rev. A **68**, 052311 (2003).
[108] Michael H. Goerz and Contributors (2022). code: qucontrol/weylchamber.
[109] Michael H. Goerz and Contributors (2022). code: JuliaQuantumControl/QuantumControlBase.jl.
[110] Yuriy Makhlin. "Nonlocal properties of two-qubit gates and mixed states, and the optimization of quantum computations". Quantum Inf. Process. **1**, 243 (2002).
[111] John R. Dormand and Pete J. Prince. "A family of embedded Runge-Kutta formulae". J. Comput. Appl. Math **6**, 19 (1980).
[112] Christopher Rackauckas and Qing Nie. "DifferentialEquations.jl – a performant and feature-rich ecosystem for solving differential equations in Julia". J. Open Res. Softw.**5** (2017).
[113] Jarrett Revels and Contributors (2022). code: JuliaCI/BenchmarkTools.jl.
[114] Yunwei Lu, Vinh San Dinh, and Jens Koch. "Increasing memory and runtime performance of GRAPE for control in large quantum systems". In Bulletin of the American Physical Society, APS March Meeting 2022, Chicago. Number 10 in Session Y41 (2022). url: meetings.aps.org/Meeting/MAR22/Session/Y41.10.
[115] Giampaolo Rodola and Contributors (2022). code: giampaolo/psutil.
[116] Will Tebbutt, Frames Catherine White, Miha Zgubic, Wessel Bruinsma, and Contributors (2022). code: JuliaDiff/FiniteDifferences.jl.
[117] Shai Machnes, Elie Assémat, David J. Tannor, and Frank K. Wilhelm. "Tunable, flexible, and efficient optimization of control pulses for practical qubits". Phys. Rev. Lett. **120**, 150401 (2018).
[118] Jens Jakob W. H. Sørensen, Mikel O. Aranburu, Till Heinzel, and Jacob F. Sherson. "Quantum optimal control in a chopped basis: Applications in control of Bose-Einstein condensates". Phys. Rev. A **98**, 022119 (2018).

[119] Dennis Lucarelli. "Quantum optimal control via gradient ascent in function space and the time-bandwidth quantum speed limit". Phys. Rev. A **97**, 062346 (2018).
[120] David L. Goodwin and Ilya Kuprov. "Modified Newton-Raphson GRAPE methods for optimal control of spin systems". J. Chem. Phys. **144**, 204107 (2016).
[121] Keno Fischer and Contributors (2022). code: JuliaDiff/Diffractor.jl.
[122] Daniel M. Reich. "Efficient characterisation and optimal control of open quantum systems. Mathematical foundations and physical applications". PhD thesis. Universität Kassel. (2015). url: d-nb.info/1073888851/34.
[123] Felix Platzer, Florian Mintert, and Andreas Buchleitner. "Optimal dynamical control of many-body entanglement". Phys. Rev. Lett. **105**, 020501 (2010).
[124] Tommaso Caneva, Tommaso Calarco, and Simone Montangero. "Entanglement-storage units". New J. Phys. **14**, 093041 (2012).
[125] Patrick Doria, Tommaso Calarco, and Simone Montangero. "Optimal control technique for many-body quantum dynamics". Phys. Rev. Lett. **106**, 190501 (2011).
[126] Tommaso Caneva, Tommaso Calarco, and Simone Montangero. "Chopped random-basis quantum optimization". Phys. Rev. A **84**, 022326 (2011).
[127] Niklas Rach, Matthias M. Müller, Tommaso Calarco, and Simone Montangero. "Dressing the chopped-random-basis optimization: A bandwidth-limited access to the trap-free landscape". Phys. Rev. A **92**, 062343 (2015).
[128] Hohjai Lee, Yuan-Chung Cheng, and Graham R. Fleming. "Coherence dynamics in photosynthesis: Protein protection of excitonic coherence". Science **316**, 1462 (2007).
[129] Susana F. Huelga and Martin B. Plenio. "Vibrations, quanta and biology". Contemp. Phys. **54**, 181 (2013).
[130] Michael H. Goerz, Mark A. Kasevich, and Vladimir S. Malinovsky. "Quantum optimal control for atomic fountain interferometry". In Proc. SPIE 11700, Optical and Quantum Sensing and Precision Metrology. (2021). url: doi.org/10.1117/12.2587002.
[131] A. Duspayev and G. Raithel. "Tractor atom interferometry". Phys. Rev. A **104**, 013307 (2021).
[132] Georg A. Raithel, Alisher Duspayev, Bineet Dash, Sebastián C. Carrasco, Michael H. Goerz, Vladan Vuletic, and Vladimir S. Malinovsky. "Principles of tractor atom interferometry". Quantum Sci. Technol. (2022).
[133] Pavel Sekatski, Michalis Skotiniotis, Janek Kołodyński, and Wolfgang Dür. "Quantum metrology with full and fast quantum control". Quantum **1**, 27 (2017).
[134] Chungwei Lin, Yanting Ma, and Dries Sels. "Optimal control for quantum metrology via Pontryagin's principle". Phys. Rev. A **103**, 052607 (2021).
[135] Sebastián C. Carrasco, Michael H. Goerz, Zeyang Li, Simone Colombo, Vladan Vuletić, and Vladimir S. Malinovsky. "Extreme spin squeezing via optimized one-axis twisting and rotations". Phys. Rev. Applied **17**, 064050 (2022).
[136] Michael H. Goerz, Sebastián C. Carrasco, and Vladimir S. Malinovsky (2022). code: ARLQCI/2022-04_semiad_paper.