



Supercharging Quantum Optimal Control with Efficient Automatic Differentiation

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

DEVCOM Army Research Lab

APS March Meeting 2022

Gradient Ascent (GRAPE)

optimum quantum control — steer quantum system in some desired way

e.g. CNOT gate:

$$|\phi_1\rangle = |00\rangle \rightarrow \text{CNOT } |00\rangle = |00\rangle = |\phi_1^{\text{tgt}}\rangle$$

$$|\phi_2\rangle = |01\rangle \rightarrow \text{CNOT } |01\rangle = |01\rangle = |\phi_2^{\text{tgt}}\rangle$$

$$|\phi_3\rangle = |10\rangle \rightarrow \text{CNOT } |10\rangle = |11\rangle = |\phi_3^{\text{tgt}}\rangle$$

$$|\phi_4\rangle = |11\rangle \rightarrow \text{CNOT } |11\rangle = |10\rangle = |\phi_4^{\text{tgt}}\rangle$$

with Hamiltonian

$$\hat{H} = \hat{H}_0 + \epsilon_1(t)\hat{H}_1 + \epsilon_2(t)\hat{H}_2$$

Gradient Ascent (GRAPE)

optimum quantum control — steer quantum system in some desired way

e.g. CNOT gate:

$$|\phi_1\rangle = |00\rangle \rightarrow \text{CNOT } |00\rangle = |00\rangle = |\phi_1^{\text{tgt}}\rangle$$

$$|\phi_2\rangle = |01\rangle \rightarrow \text{CNOT } |01\rangle = |01\rangle = |\phi_2^{\text{tgt}}\rangle$$

$$|\phi_3\rangle = |10\rangle \rightarrow \text{CNOT } |10\rangle = |11\rangle = |\phi_3^{\text{tgt}}\rangle$$

$$|\phi_4\rangle = |11\rangle \rightarrow \text{CNOT } |11\rangle = |10\rangle = |\phi_4^{\text{tgt}}\rangle$$

with Hamiltonian

$$\hat{H} = \hat{H}_0 + \epsilon_1(t)\hat{H}_1 + \epsilon_2(t)\hat{H}_2$$

Iteratively minimize functional, e.g.

$$J = 1 - \left| \frac{1}{4} \sum_{k=1}^4 \langle \phi_k^{\text{tgt}} | \phi_k(T) \rangle \right|^2$$

Gradient Ascent (GRAPE)

optimum quantum control — steer quantum system in some desired way

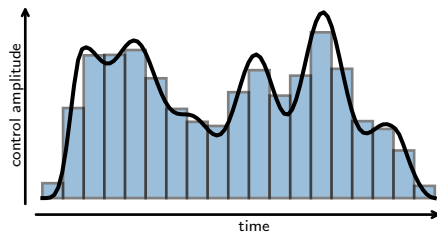
e.g. CNOT gate:

$$|\phi_1\rangle = |00\rangle \rightarrow \text{CNOT } |00\rangle = |00\rangle = |\phi_1^{\text{tgt}}\rangle$$

$$|\phi_2\rangle = |01\rangle \rightarrow \text{CNOT } |01\rangle = |01\rangle = |\phi_2^{\text{tgt}}\rangle$$

$$|\phi_3\rangle = |10\rangle \rightarrow \text{CNOT } |10\rangle = |11\rangle = |\phi_3^{\text{tgt}}\rangle$$

$$|\phi_4\rangle = |11\rangle \rightarrow \text{CNOT } |11\rangle = |10\rangle = |\phi_4^{\text{tgt}}\rangle$$



with Hamiltonian

$$\hat{H} = \hat{H}_0 + \epsilon_1(t)\hat{H}_1 + \epsilon_2(t)\hat{H}_2$$

Iteratively minimize functional, e.g.

$$J = 1 - \left| \frac{1}{4} \sum_{k=1}^4 \langle \phi_k^{\text{tgt}} | \phi_k(T) \rangle \right|^2$$

Gradient descent: [Khaneja *et al.*, JMR 172, 296 (2005)]

gradient $\frac{\partial J}{\partial \epsilon_{ln}}$ with $\epsilon_{ln} = \epsilon_l(t_n)$

$\Rightarrow$ feed into solver (LBFGS)

[de Fouquières *et al.*, JMR 212, 412 (2011)]

Evaluating Gradients

$$\text{gradient } \frac{\partial J}{\partial \epsilon_{ln}} \text{ for } J = 1 - \left| \frac{1}{4} \sum_{k=1}^4 \underbrace{\langle \phi_k^{\text{tgt}} | \phi_k(T) \rangle}_{=\tau_k} \right|^2$$

Evaluating Gradients

$$\text{gradient } \frac{\partial J}{\partial \epsilon_{ln}} \text{ for } J = 1 - \left| \frac{1}{4} \sum_{k=1}^4 \underbrace{\langle \phi_k^{\text{tgt}} | \phi_k(T) \rangle}_{=\tau_k} \right|^2$$

$$\begin{aligned} \frac{\partial \tau_k}{\partial \epsilon_{ln}} &= \frac{\partial}{\partial \epsilon_{ln}} \langle \phi_k^{\text{tgt}} | \phi_k(T) \rangle = \frac{\partial}{\partial \epsilon_{ln}} \langle \phi_k^{\text{tgt}} | \hat{U}_{N_T} \dots \hat{U}_n \dots \hat{U}_1 | \phi_k \rangle \\ &= \underbrace{\langle \phi_k^{\text{tgt}} | \hat{U}_{N_T} \dots \hat{U}_{n+1}}_{\langle \chi_k(t_{n+1}) |} \frac{\partial \hat{U}_n}{\partial \epsilon_{ln}} \underbrace{\hat{U}_{n-1} \dots \hat{U}_1 | \phi_k \rangle}_{| \phi_k(t_n) \rangle} \end{aligned}$$

Evaluating Gradients

$$\text{gradient } \frac{\partial J}{\partial \epsilon_{ln}} \text{ for } J = 1 - \left| \frac{1}{4} \sum_{k=1}^4 \underbrace{\langle \phi_k^{\text{tgt}} | \phi_k(T) \rangle}_{=\tau_k} \right|^2$$

$$\begin{aligned} \frac{\partial \tau_k}{\partial \epsilon_{ln}} &= \frac{\partial}{\partial \epsilon_{ln}} \langle \phi_k^{\text{tgt}} | \phi_k(T) \rangle = \frac{\partial}{\partial \epsilon_{ln}} \langle \phi_k^{\text{tgt}} | \hat{U}_{N_T} \dots \hat{U}_n \dots \hat{U}_1 | \phi_k \rangle \\ &= \underbrace{\langle \phi_k^{\text{tgt}} | \hat{U}_{N_T} \dots \hat{U}_{n+1}}_{\langle \chi_k(t_{n+1}) |} \frac{\partial \hat{U}_n}{\partial \epsilon_{ln}} \underbrace{\hat{U}_{n-1} \dots \hat{U}_1 | \phi_k \rangle}_{| \phi_k(t_n) \rangle} \end{aligned}$$

$$\frac{\partial J}{\partial \epsilon_{ln}} = -\frac{1}{N^2} \sum_{k,k'=1}^N \left[\frac{\partial \tau_{k'}^*}{\partial \epsilon_{ln}} \tau_k + \tau_{k'}^* \frac{\partial \tau_k}{\partial \epsilon_{ln}} \right] = -\frac{2}{N^2} \text{Re} \sum_{k,k'=1}^N \tau_{k'}^* \frac{\partial \tau_k}{\partial \epsilon_{ln}}$$

Evaluating Gradients

$$\tilde{G} = \begin{pmatrix} \hat{H}(t) & 0 & \dots & 0 & \hat{H}_1 \\ 0 & \hat{H}(t) & \dots & 0 & \hat{H}_2 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & \hat{H}(t) & \hat{H}_L \\ 0 & 0 & \dots & 0 & \hat{H}(t) \end{pmatrix}$$

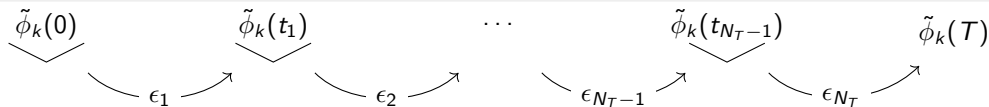
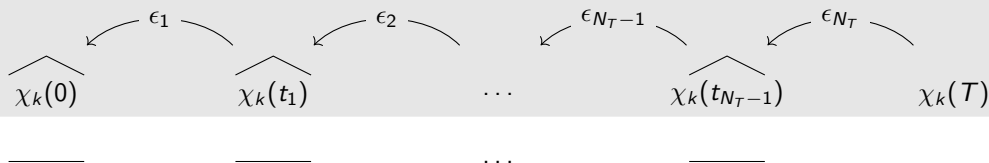
$$e^{-i\tilde{G}dt} \underbrace{\begin{pmatrix} 0 \\ \vdots \\ 0 \\ |\phi\rangle \end{pmatrix}}_{|\tilde{\phi}\rangle} = \begin{pmatrix} \frac{\partial \hat{U}_n}{\partial \epsilon_{1,n}} |\phi\rangle \\ \vdots \\ \frac{\partial \hat{U}_n}{\partial \epsilon_{L,n}} |\phi\rangle \\ \hat{U}_n |\phi\rangle \end{pmatrix}$$

[Van Loan. IEEE Trans. Automat. Contr. 23, 395 (1978)]

[Floether *et al.* NJP. 14, 073023 (2012)]

Evaluating Gradients

① backward-propagation and storage



② forward-propagation of extended state + gradient

evaluates $\frac{\partial \tau_k}{\partial \epsilon_{In}} = \frac{\partial}{\partial \epsilon_{In}} \langle \phi_k^{\text{tgt}} | \phi_k(T) \rangle$

Functionals

Most GRAPE implementations have e.g. $J = 1 - \left| \frac{1}{4} \sum_{k=1}^4 \langle \phi_k^{\text{tgt}} | \phi_k(\mathcal{T}) \rangle \right|^2$ hard-coded

Functionals

Most GRAPE implementations have e.g. $J = 1 - \left| \frac{1}{4} \sum_{k=1}^4 \langle \phi_k^{\text{tgt}} | \phi_k(\mathcal{T}) \rangle \right|^2$ hard-coded

What about other functionals?

Functionals

Most GRAPE implementations have e.g. $J = 1 - \left| \frac{1}{4} \sum_{k=1}^4 \langle \phi_k^{\text{tgt}} | \phi_k(\mathcal{T}) \rangle \right|^2$ hard-coded

What about other functionals?

What if we cannot calculate the derivative on paper?

Functionals

Most GRAPE implementations have e.g. $J = 1 - \left| \frac{1}{4} \sum_{k=1}^4 \langle \phi_k^{\text{tgt}} | \phi_k(\mathcal{T}) \rangle \right|^2$ hard-coded

What about other functionals?

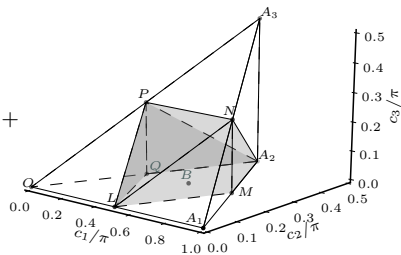
What if we cannot calculate the derivative on paper?

① Perfect Entanglers Functional

Find a two-qubit gate with maximum entangling power

$$F_{PE} = \left(\frac{1}{\det U_B} \right) \left(\frac{1}{4} (\text{tr}^2[U_B^T U_B] - \text{tr}[U_B^T U_B U_B^T U_B]) \right) \left(\frac{1}{16} \text{Re}^2[\text{tr}[U_B^T U_B]] \right) + \\ + \left(\frac{2}{\det U_B} \right) \left(\frac{1}{4} (\text{tr}^2[U_B^T U_B] - \text{tr}[U_B^T U_B U_B^T U_B]) \right) \left(\frac{1}{16} \text{Im}^2[\text{tr}[U_B^T U_B]] \right) \\ \left(\frac{1}{16} \text{Re}[\text{tr}^2[U_B^T U_B]] \right)$$

U_B : projection into logical subspace, in Bell basis



[Watts *et al.*, PRA 91, 062306 (2015)]

[Goerz *et al.*, PRA 91, 062307 (2015)]

Functionals

Most GRAPE implementations have e.g. $J = 1 - \left| \frac{1}{4} \sum_{k=1}^4 \langle \phi_k^{\text{tgt}} | \phi_k(\mathcal{T}) \rangle \right|^2$ hard-coded

What about other functionals?

What if we cannot calculate the derivative on paper?

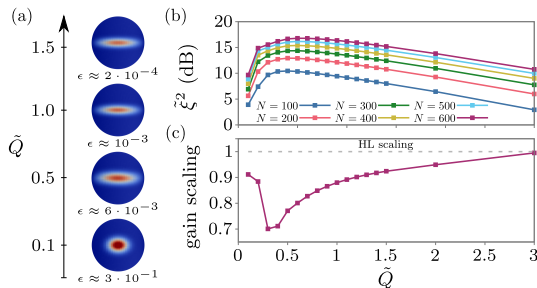
② Quantum Fisher Information

Find a state that maximizes metrological gain

$$F(\hat{\rho}_\theta) = \sum_{i \neq j} \frac{1(p_i - p_j)^2}{p_i + p_j} \left| \langle \phi_i | \hat{J}_\theta | \phi_j \rangle \right|^2$$

where p_i , $|\phi_i\rangle$ eigenvalues/eigenvectors of $\hat{\rho}$

[Ma et al., Phys. Rep. 509, 89 (2011)]

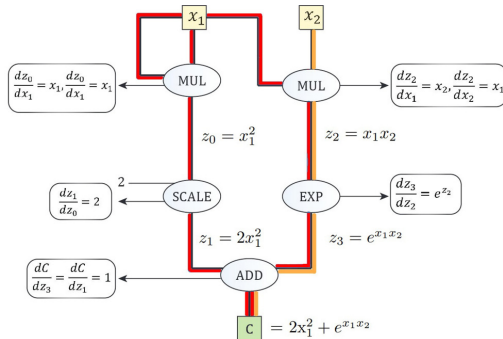


[Carrasco et al., arXiv 2201.01744 (2022)]

Session N39 (Wed 11:30)

Automatic Differentiation

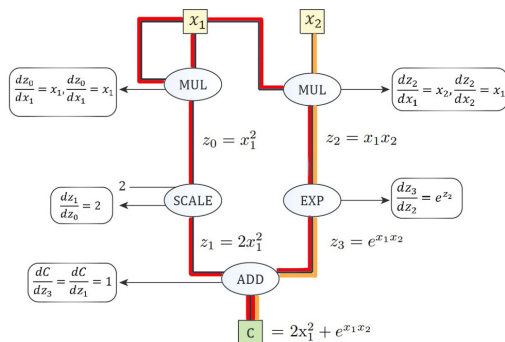
To a computer, *everything* is analytic!



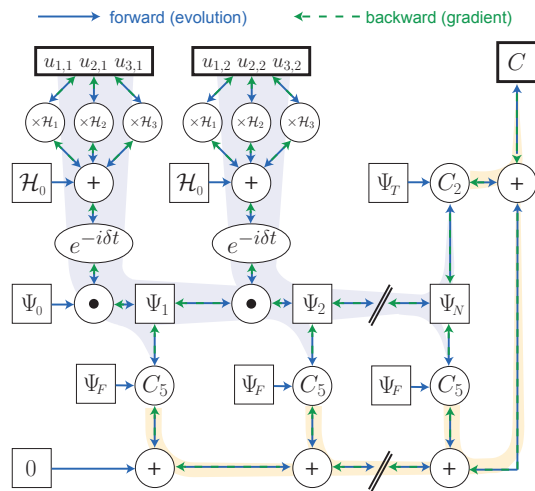
[Abdelhafez et al. PRA 99, 052327 (2019)]

Automatic Differentiation

To a computer, *everything* is analytic!



[Abdelhafez *et al.* PRA 99, 052327 (2019)]

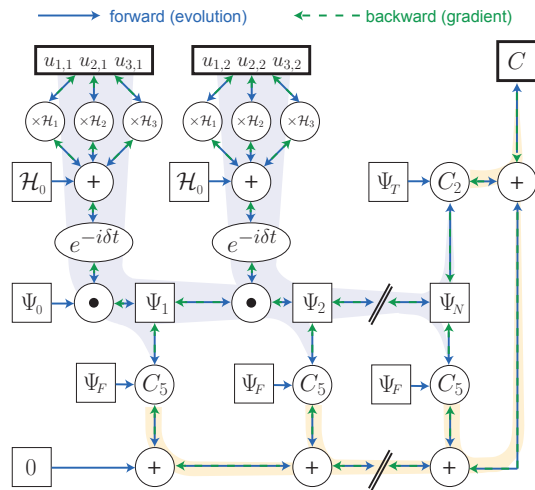


[Leung *et al.* PRA 95, 042318 (2017)]

Automatic Differentiation

Advantages

- Can optimize any functional



[Leung *et al.* PRA 95, 042318 (2017)]

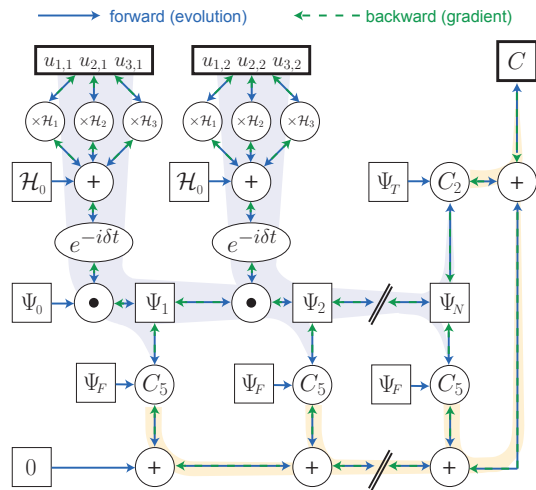
Automatic Differentiation

Advantages

- Can optimize any functional

Disadvantages

- Little code reuse
- Inefficient propagators (matrix exponentiation, no in-place operations)
- Extremely large memory overhead for storing gradients in graph



[Leung *et al.* PRA 95, 042318 (2017)]

Automatic Differentiation

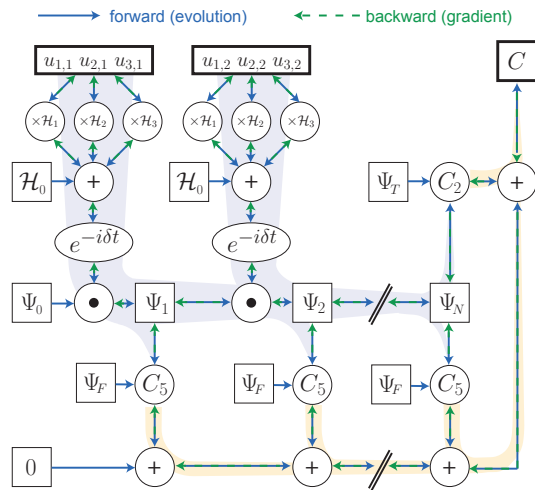
Advantages

- Can optimize any functional

Disadvantages

- Little code reuse
- Inefficient propagators (matrix exponentiation, no in-place operations)
- Extremely large memory overhead for storing gradients in graph

Can we keep AD advantage while building on existing GRAPE implementations?



[Leung *et al.* PRA 95, 042318 (2017)]

Semi-Automatic Differentiation

Idea: Help AD with the chain rule — push numerical effort into “inner” derivative

Semi-Automatic Differentiation

Idea: Help AD with the chain rule — push numerical effort into “inner” derivative

①

$$\frac{\partial J(\{\tau_k\})}{\partial \epsilon_{In}} = \underbrace{\frac{\partial J}{\partial \tau_k}}_{\text{AD}} \underbrace{\frac{\partial \tau_k}{\partial \epsilon_{In}}}_{\text{GRAPE}} \quad \text{with} \quad \tau_k = \langle \phi_k^{\text{tgt}} | \phi_k(T) \rangle \quad \Rightarrow \text{standard functionals}$$

Semi-Automatic Differentiation

Idea: Help AD with the chain rule — push numerical effort into “inner” derivative

①

$$\frac{\partial J(\{\tau_k\})}{\partial \epsilon_{In}} = \underbrace{\frac{\partial J}{\partial \tau_k}}_{\text{AD}} \underbrace{\frac{\partial \tau_k}{\partial \epsilon_{In}}}_{\text{GRAPE}} \quad \text{with} \quad \tau_k = \langle \phi_k^{\text{tgt}} | \phi_k(T) \rangle \quad \Rightarrow \text{standard functionals}$$

②

$$\frac{\partial J(\{\tau_{k'k}\})}{\partial \epsilon_{In}} = \underbrace{\frac{\partial J}{\partial \tau_{k'k}}}_{\text{AD}} \underbrace{\frac{\partial \tau_{k'k}}{\partial \epsilon_{In}}}_{\text{GRAPE}^*} \quad \text{with} \quad \tau_{k'k} = \langle \phi_{k'}^{\text{tgt}} | \phi_k(T) \rangle \quad \Rightarrow \text{functionals of gates (PE)}$$

Semi-Automatic Differentiation

Idea: Help AD with the chain rule — push numerical effort into “inner” derivative

①

$$\frac{\partial J(\{\tau_k\})}{\partial \epsilon_{ln}} = \underbrace{\frac{\partial J}{\partial \tau_k}}_{\text{AD}} \underbrace{\frac{\partial \tau_k}{\partial \epsilon_{ln}}}_{\text{GRAPE}} \quad \text{with} \quad \tau_k = \langle \phi_k^{\text{tgt}} | \phi_k(T) \rangle \quad \Rightarrow \text{standard functionals}$$

②

$$\frac{\partial J(\{\tau_{k'k}\})}{\partial \epsilon_{ln}} = \underbrace{\frac{\partial J}{\partial \tau_{k'k}}}_{\text{AD}} \underbrace{\frac{\partial \tau_{k'k}}{\partial \epsilon_{ln}}}_{\text{GRAPE}^*} \quad \text{with} \quad \tau_{k'k} = \langle \phi_{k'}^{\text{tgt}} | \phi_k(T) \rangle \quad \Rightarrow \text{functionals of gates (PE)}$$

③

$$\frac{\partial J(\{|\Psi_k\rangle\})}{\partial \epsilon_{ln}} = 2\text{Re} \left[\underbrace{\frac{\partial J}{\partial |\Psi_k\rangle}}_{\text{AD}} \frac{\partial |\Psi_k\rangle}{\partial \epsilon_{ln}} \right] = 2\text{Re} \left\langle \chi_k \left| \frac{\partial \Psi_k}{\partial \epsilon_{ln}} \right. \right\rangle = 2\text{Re} \underbrace{\frac{\partial}{\partial \epsilon_{ln}} \langle \chi_k | \Psi_k \rangle}_{\text{GRAPE}} \quad \Rightarrow \text{arbitrary functionals (QFI)}$$

GRAPE.jl



The screenshot shows a web browser window displaying the GitHub repository for GRAPE.jl. The browser's address bar shows the URL `github.com/JuliaQuantumControl/GRAPE.jl`. The repository's README file is open, showing the title **GRAPE.jl** and a header section with badges for version (v0.1.1), documentation (stable, dev), CI status (passing), and code coverage (81%). The README text describes GRAPE as an implementation of (second-order) GRadiant Ascent Pulse Engineering (GRAPE) extended with automatic differentiation, part of the `QuantumControl.jl` package and the JuliaQuantumControl organization. An **Installation** section follows, stating that the GRAPE package should be installed as part of `QuantumControl.jl`. A code block at the bottom shows the command `pkg> add QuantumControl`.

GitHub - JuliaQuantumControl/ x +

← → ↻ `github.com/JuliaQuantumControl/GRAPE.jl` 📄 ☆ ⚙️ 👤 ⋮

☰ README.md

GRAPE.jl

Mar 2022 `v0.1.1` docs `stable` docs `dev` CI `passing` `codecov` 81%

Implementation of (second-order) GRadiant Ascent Pulse Engineering (GRAPE) extended with automatic differentiation. Part of `QuantumControl.jl` and the JuliaQuantumControl organization.

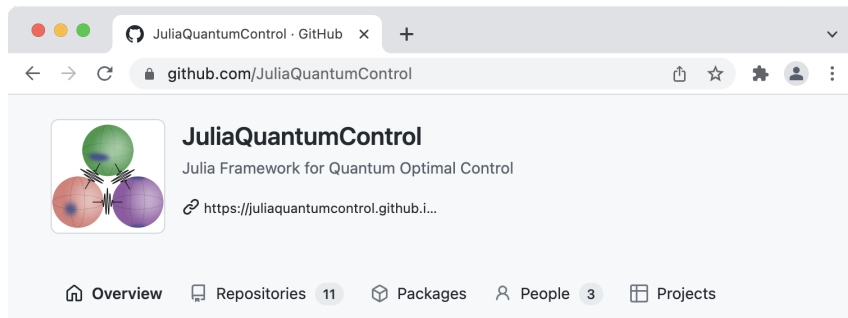
Installation

For normal usage, the GRAPE package should be installed as part of `QuantumControl.jl`:

```
pkg> add QuantumControl
```



JuliaQuantumControl



JuliaQuantumControl · GitHub

github.com/JuliaQuantumControl

JuliaQuantumControl
Julia Framework for Quantum Optimal Control
[https://juliaquantumcontrol.github.io/...](https://juliaquantumcontrol.github.io/)

Overview Repositories 11 Packages People 3 Projects



README.md

🔗 A Julia Framework for Quantum Optimal Control.

The [JuliaQuantumControl](#) organization collects packages implementing a comprehensive collection of methods of open-loop quantum optimal control.

Conclusions & Outlook

Summary

- AD enables optimization of *arbitrary* functionals
- AD can be made efficient with “semi-AD”
- All functionals can use “standard” GRAPE as main numerical component

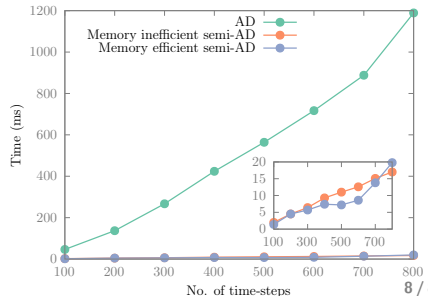
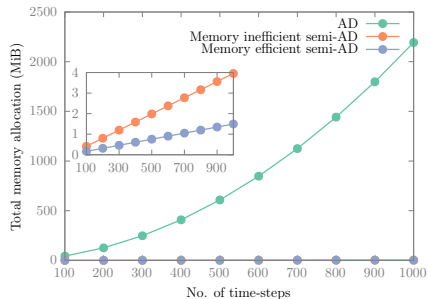
Conclusions & Outlook

Summary

- AD enables optimization of *arbitrary* functionals
- AD can be made efficient with “semi-AD”
- All functionals can use “standard” GRAPE as main numerical component

Outlook

- Support all semi-AD variations in GRAPE.jl
- Usage examples —
gate optimization & metrology
- Benchmark memory & runtime advantage



Conclusions & Outlook

Summary

- AD enables optimization of *arbitrary* functionals
- AD can be made efficient with “semi-AD”
- All functionals can use “standard” GRAPE as main numerical component

Outlook

- Support all semi-AD variations in GRAPE.jl
- Usage examples —
gate optimization & metrology
- Benchmark memory & runtime advantage

Thank You!

