Modernizing the Quantum Control Stack with QuantumControl.jl

Michael H. Goerz

DEVCOM Army Research Lab

Quantum Control Workshop, Berlin, May 22, 2024

JuliaQuantumControl

github.com

JuliaQuantumControl

Julia Framework for Quantum Optimal Control

20 followers https://juliaquantumcontrol.github.i...

Overview | Repositories 15 | Discussions | Projects | Packages | People 3

README.md

A Julia Framework for Quantum Optimal Control.

docs stable docs dev

The JuliaQuantumControl organization collects packages implementing a comprehensive collection of methods of open-loop quantum optimal control.

Quantum optimal control theory attempts to steer a quantum system in some desired way by finding optimal control parameters or control fields inside the system Hamiltonian or Liouvillian. Typical control tasks are the preparation of a specific quantum state or the realization of a logical gate in a quantum computer. Thus, quantum control theory is a critical part of realizing quantum technologies, at the lowest level. Numerical methods of *open-loop* quantum control (methods that do not involve measurement feedback from a physical quantum device) such as Krotov's method and GRAPE address the control problem by simulating the dynamics of the system and then iteratively improving the value of a functional that encodes the desired outcome.

People

Top languages

Julia Makefile

Most used topics

julia quantum grape optimal-control quantum-computing

JuliaQuantumControl

github.com

Packages

Package	Version	CI Status	Coverage	Description
⭐ QuantumPropagators.jl	May 2023 v0.6.0	CI passing	codecov 90%	Simulate the time evolution of quantum systems (docs)
QuantumControlBase.jl	May 2023 v0.8.3	CI passing	codecov 89%	Shared methods and data structures (docs)
QuantumGradientGenerators.jl	May 2023 v0.1.2	CI passing	codecov 81%	Dynamic Gradients for Quantum Control (docs)
Krotov.jl	Mar 2023 v0.5.3	CI passing	codecov 90%	Krotov's method of optimal control (docs)
GRAPE.jl	Mar 2023 v0.5.4	CI passing	codecov 79%	Gradient Ascent Pulse Engineering method (docs)
TwoQubitWeylChamber.jl	Mar 2023 v0.1.1	CI passing	codecov 97%	Optimizing two-qubit gates in the Weyl chamber (docs)
QuantumControlTestUtils.jl	May 2023 v0.1.5	CI passing		Tools for testing and benchmarking (docs)
⭐ QuantumControl.jl	May 2023 v0.8.0	CI passing	codecov 78%	Framework for Quantum Dynamics and Control (docs)

Why Julia?

- Flexibility
- Performance
- Expressiveness

Multiple Dispatch

Julia's secret sauce: "multiple dispatch"

- Function name has table of "methods" (signatures)
- Pick the method that most narrowly matches signature
- Adding methods *dynamically* recompiles anything calling the function, if necessary

See video: "The Unreasonable Effectiveness of Multiple Dispatch"

Multiple Dispatch

julia

```
julia> LinearAlgebra.mul!
mul! (generic function with 31 methods)

julia> methods(LinearAlgebra.mul!)
# 31 methods for generic function "mul!" from LinearAlgebra:
  [1] mul!(A::LinearAlgebra.AbstractTriangular, B::LinearAlgebra.AbstractTriangular, C::Number, alpha::Number, beta::Number)
     @ ~/.julia/juliaup/julia-1.10.2+0.aarch64.apple.darwin14/share/julia/stdlib/v1.10/LinearAlgebra/src/triangular.jl:467
  [2] mul!(A::LinearAlgebra.AbstractTriangular, B::Number, C::LinearAlgebra.AbstractTriangular, alpha::Number, beta::Number)
     @ ~/.julia/juliaup/julia-1.10.2+0.aarch64.apple.darwin14/share/julia/stdlib/v1.10/LinearAlgebra/src/triangular.jl:469
  [3] mul!(C::AbstractVecOrMat{T}, Q::LinearAlgebra.AbstractQ{T}, B::Union{LinearAlgebra.AbstractQ, AbstractVecOrMat}) where T
     @ ~/.julia/juliaup/julia-1.10.2+0.aarch64.apple.darwin14/share/julia/stdlib/v1.10/LinearAlgebra/src/abstractq.jl:200
  [4] mul!(C::AbstractMatrix, A::LinearAlgebra.AbstractTriangular, B::LinearAlgebra.AbstractTriangular)
     @ ~/.julia/juliaup/julia-1.10.2+0.aarch64.apple.darwin14/share/julia/stdlib/v1.10/LinearAlgebra/src/triangular.jl:693
  [5] mul!(C::AbstractMatrix, A::LinearAlgebra.AbstractTriangular, B::LinearAlgebra.AbstractTriangular, alpha::Number, beta::N
umber)
     @ ~/.julia/juliaup/julia-1.10.2+0.aarch64.apple.darwin14/share/julia/stdlib/v1.10/LinearAlgebra/src/triangular.jl:736
  [6] mul!(C::AbstractMatrix, A::LinearAlgebra.AbstractTriangular, B::Union{LinearAlgebra.Bidiagonal, LinearAlgebra.Diagonal,
LinearAlgebra.SymTridiagonal, LinearAlgebra.Tridiagonal})
     @ ~/.julia/juliaup/julia-1.10.2+0.aarch64.apple.darwin14/share/julia/stdlib/v1.10/LinearAlgebra/src/special.jl:111
  [7] mul!(C::AbstractMatrix, A::Union{LinearAlgebra.Bidiagonal, LinearAlgebra.Diagonal, LinearAlgebra.SymTridiagonal, LinearA
lgebra.Tridiagonal}, B::LinearAlgebra.AbstractTriangular)
     @ ~/.julia/juliaup/julia-1.10.2+0.aarch64.apple.darwin14/share/julia/stdlib/v1.10/LinearAlgebra/src/special.jl:112
  [8] mul!(C::AbstractMatrix, A::LinearAlgebra.AbstractTriangular, B::AbstractMatrix)
     @ ~/.julia/juliaup/julia-1.10.2+0.aarch64.apple.darwin14/share/julia/stdlib/v1.10/LinearAlgebra/src/triangular.jl:691
  [9] mul!(C::AbstractMatrix, A::AbstractMatrix, B::LinearAlgebra.AbstractTriangular)
     @ ~/.julia/juliaup/julia-1.10.2+0.aarch64.apple.darwin14/share/julia/stdlib/v1.10/LinearAlgebra/src/triangular.jl:692
 [10] mul!(C::AbstractVecOrMat, A::LinearAlgebra.AbstractTriangular, B::AbstractVector)
     @ ~/.julia/juliaup/julia-1.10.2+0.aarch64.apple.darwin14/share/julia/stdlib/v1.10/LinearAlgebra/src/triangular.jl:690
```

Multiple Dispatch

Define high-level interfaces

Control Problem and Trajectories

juliaquantumcontrol.github.io

QuantumControlBase.ControlProblem — Type

A full control problem with multiple trajectories.

```
ControlProblem(
    trajectories,
    tlist;
    kwargs...
)
```

The `trajectories` are a list of `Trajectory` instances, each defining an initial state and a dynamical generator for the evolution of that state. Usually, the trajectory will also include a target state (see `Trajectory`) and possibly a weight. The `trajectories` may also be given together with `tlist` as a mandatory keyword argument.

The `tlist` is the time grid on which the time evolution of the initial states of each trajectory should be propagated. It may also be given as a (mandatory) keyword argument.

The remaining `kwargs` are keyword arguments that are passed directly to the optimal control method. These typically include e.g. the optimization functional.

Dynamical Generator

juliaquantumcontrol.github.io

Glossary

Generator — Dynamical generator (Hamiltonian / Liouvillian) for the time evolution of a state, i.e., the right-hand-side of the equation of motion (up to a factor of i) such that $|\Psi(t+dt)\rangle = e^{-i\hat{H}dt}|\Psi(t)\rangle$ in the infinitesimal limit. We use the symbols G, $\hat{H}$, or L, depending on the context (general, Hamiltonian, Liouvillian). Examples for supported forms a Hamiltonian are the following, from the most general case to simplest and most common case of linear controls,

$$\hat{H} = \overbrace{\hat{H}_0}^{\text{drift term}} + \sum_l \overbrace{\hat{H}_l(\{\epsilon_{l'}(t)\}, t)}^{\text{control term}} \tag{G1}$$

$$\hat{H} = \hat{H}_0 + \sum_l \overbrace{a_l(\{\underbrace{\epsilon_{l'}(t)}_{\text{control function}}\}, t)}^{\text{control amplitude}} \hat{H}_l \tag{G2}$$

$$\hat{H} = \hat{H}_0 + \sum_l \overbrace{\epsilon_l(t)} \underbrace{\hat{H}_l}_{\text{control operator}} \tag{G3}$$

Generator Interface

juliaquantumcontrol.github.io

```
@test check_generator(
    generator; state, tlist,
    for_mutable_operator=true, for_immutable_operator=true,
    for_mutable_state=true, for_immutable_state=true,
    for_pwc=true, for_time_continuous=false,
    for_expval=true, for_parameterization=false,
    atol=1e-14, quiet=false)
```

verifies the given `generator`:

- `get_controls(generator)` must be defined and return a tuple
- all controls returned by `get_controls(generator)` must pass `check_control`
- `substitute(generator, replacements)` must be defined
- If `generator` is a `Generator` instance, all elements of `generator.amplitudes` must pass `check_amplitude` with `for_parameterization`.

If `for_pwc` (default):

- `evaluate(generator, tlist, n)` must return a valid operator (`check_operator`), with forwarded keyword arguments (including `for_expval`)

Generator Interface

juliaquantumcontrol.github.io

If `for_pwc` (default):

- `evaluate(generator, tlist, n)` must return a valid operator (`check_operator`), with forwarded keyword arguments (including `for_expval`)
- If `for_mutable_operator`, `evaluate!(op, generator, tlist, n)` must be defined

If `for_time_continuous`:

- `evaluate(generator, t)` must return a valid operator (`check_operator`), with forwarded keyword arguments (including `for_expval`)
- If `for_mutable_operator`, `evaluate!(op, generator, t)` must be defined

If `for_parameterization` (may require the `RecursiveArrayTools` package to be loaded):

- `get_parameters(generator)` must be defined and return a vector of floats. Mutating that vector must mutate the controls inside the `generator`.

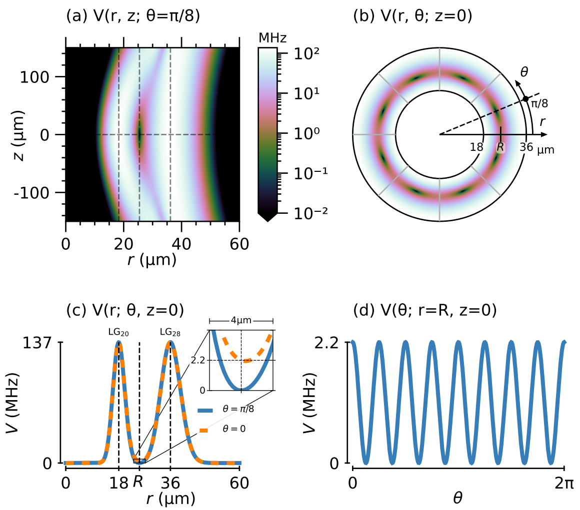
The function returns `true` for a valid generator and `false` for an invalid generator. Unless `quiet=true`, it will log an error to indicate which of the conditions failed.

source

Multiple Dispatch

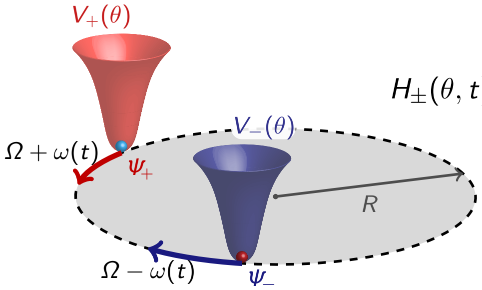
Define low-level problem-specific data structures

Rotating Tractor Interferometer



— Dash, Goerz *et al.* AVS Quantum Sci. 6, 014407 (2023)

Rotating Tractor Interferometer



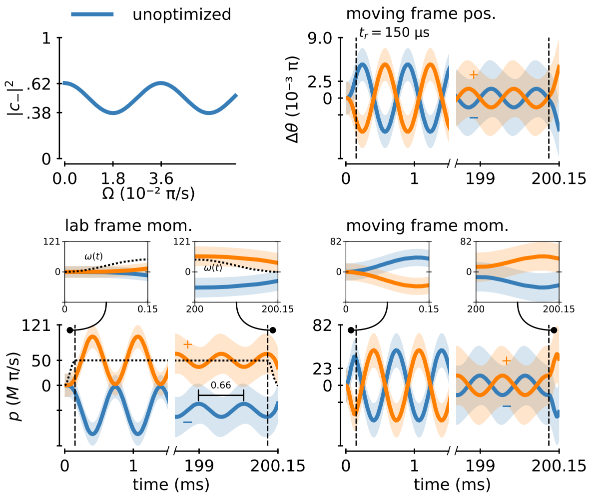
$$H_\pm(\theta, t) = -\frac{\hbar^2}{2M}\frac{\partial^2}{\partial\theta^2} + V_0 \cos\left(m(\theta + \phi_\pm(t))\right)$$

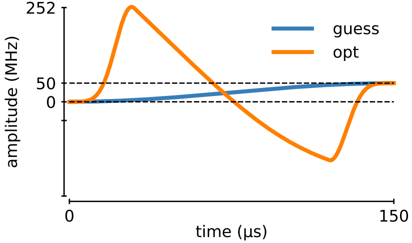
In co-moving frame:

$$\tilde{H}_\pm(t) = -\frac{\hbar^2}{2M}\frac{\partial^2}{\partial\theta^2} + V_0 \cos\left(m\theta\right) - i\hbar\omega_\pm(t)\frac{\partial}{\partial\theta}$$

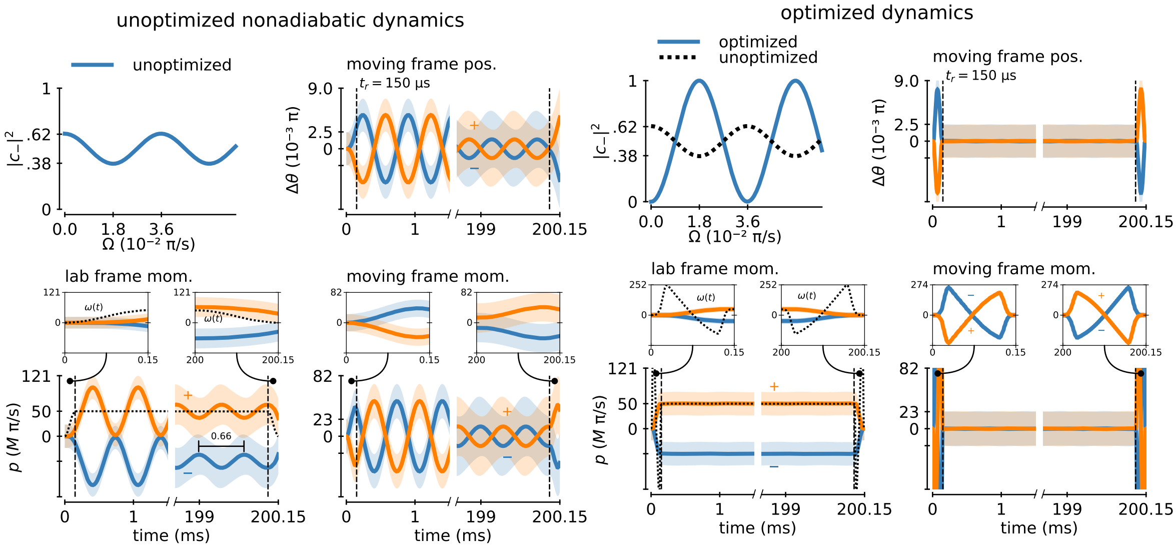
Rotating Tractor Interferometer – Optimization

unoptimized nonadiabatic dynamics





Rotating Tractor Interferometer – Optimization



Project-Specific Data Structures

Rotating TAI Implementation

```julia
struct SplitGenerator
    T  # (potentially) time-dependent
    V  # time-dependent
    to_p!::Function
    to_x!::Function
end

function get_controls(gen::SplitGenerator)
    if !isnothing(gen.T) && !isnothing(gen.V)
        return (get_controls(gen.T)..., get_controls(gen.V)...
    elseif isnothing(gen.T) && !isnothing(gen.V)
        return get_controls(gen.V)
    elseif !isnothing(gen.T) && isnothing(gen.V)
```

rotating_tai.jl

```julia
struct SplitOperator{TT,TV}
    T::TT
    V::TV
    to_p!::Function # coord to momentum
    to_x!::Function # momentum to coord
    function SplitOperator(T, V, to_p!, to_x!)
        T::Union{Nothing,Diagonal{Float64,Vector{Float64}}}
        V::Union{Nothing,Diagonal{Float64,Vector{Float64}}}
        # ishermitian depends on these type-asserts
        new{typeof(T),typeof(V)}(T, V, to_p!, to_x!)
    end
end
```

rotating_tai.jl

```julia
function LinearAlgebra.mul!(C, A::SplitOperator, B, α, β)
    # |C⟩ = β |C⟩ + α Â |B⟩ = (β |C⟩ + α V̂ |B⟩) + α T̂ |B⟩
    mul!(C, A.V, B, α, β)
    A.to_p!(B)
    A.to_p!(C)
    mul!(C, A.T, B, α, true)
    A.to_x!(B)
    A.to_x!(C)
    return C
end
```

N 8% ¶ 38/444: 1⟩ "α rotating_tai.jl ⟨julia⟨master

QuantumControl.jl is not a modeling framework!

docs.qojulia.org

QuantumOptics.jl

Search docs

Quantum systems

- Introduction
- Spin
- Fock space
- N-Level
- Particle
  - States
  - Operators
  - Additional functions

Quantum systems / Particle

Particle

```
xmin = -2.
xmax = 4.
N = 10
b_position = PositionBasis(xmin, xmax, N)
b_momentum = MomentumBasis(b_position)

x0 = 1.2
p0 = 0.4
sigma = 0.2
psi = gaussianstate(b_position, x0, p0, sigma)

x = position(b_position)
p = momentum(b_position)
```

For particles QuantumOptics.jl provides two different choices - either the calculations can be done in real space or they can be done in momentum space by using `PositionBasis` or `MomentumBasis` respectively. The definition of these two bases types is:

Flexibility

Tie in to modern techniques: automatic differentiation

Automatic differentiation (AD)

- Just do the propagation (evaluate the functional)
- Let the computer calculate the derivative $\partial J/\partial\epsilon_{nl}$

— Leung *et al.* Phys. Rev. A 95, 042318 (2017)

— Abdelhafez *et al.*, Phys. Rev. A 99, 052327 (2019)

— Schäfer, *et al.* Mach. Learn.: Sci. Technol. 1, 035009 (2020)

— Abdelhafez *et al.* Phys. Rev. A 101, 022321 (2020)

Automatic differentiation (AD)

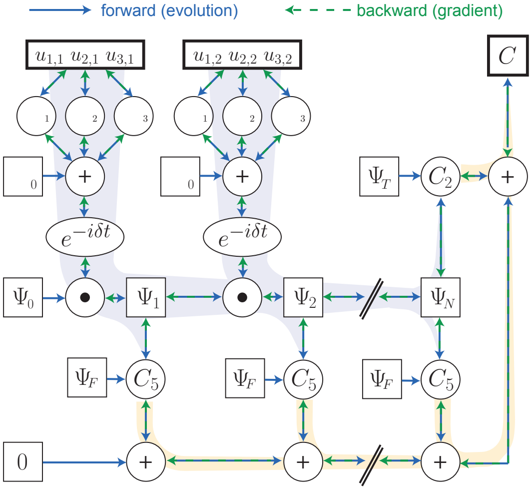


Fig. 2 in Leung *et al.* Phys. Rev. A 95, 042318 (2017)

Semi-Automatic Differentiation

Quantum 6, 871 (2022) — arXiv:2205.15044

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 differenti-

Funding

DEVCOM Army Research Laboratory, Cooperative Agreement Numbers W911NF-16-2-0147, W911NF-21-2-0037; DIRA-TRC No. DTR19-CI-019

Semi-Automatic Differentiation

$$
\begin{aligned}
\nabla J(\{\epsilon_{nl}\}) &= \frac{\partial}{\partial_{\epsilon_{nl}}} J_T(\{|\Psi_k(T)\rangle\}) + \dots \\
&= 2\mathrm{Re} \sum_k \underbrace{\frac{\partial J_T}{\partial\,|\Psi_k(T)\rangle}}_{\equiv\langle\chi_k(T)|} \frac{\partial\,|\Psi_k(T)\rangle}{\partial\epsilon_{nl}}; \qquad |\chi_k(T)\rangle = \frac{\partial J_T}{\partial\,\langle\Psi_k(T)|} \\
&= 2\mathrm{Re} \sum_k \frac{\partial}{\partial\epsilon_{nl}} \langle\chi_k(T)|\Psi_k(T)\rangle \\
&= 2\mathrm{Re} \sum_k \frac{\partial}{\partial\epsilon_{nl}} \langle\chi_k(T)|\hat{\mathrm{U}}_N \dots \hat{\mathrm{U}}_{n+1}\hat{\mathrm{U}}_n\hat{\mathrm{U}}_{n-1} \dots \hat{\mathrm{U}}_1|\Psi_k(t=0)\rangle \\
&= 2\mathrm{Re} \sum_k \underbrace{\left\langle \chi_k(T) \middle| \hat{\mathrm{U}}_N \dots \hat{\mathrm{U}}_{n+1} \frac{\partial\hat{\mathrm{U}}_n}{\partial\epsilon_{nl}}\right.}_{\text{backward propagation}} \underbrace{\left.\hat{\mathrm{U}}_{n-1} \dots \hat{\mathrm{U}}_1 \middle| \Psi_k(t=0) \right\rangle}_{\text{forward propagation}}
\end{aligned}
$$

Aside: Wirtinger derivatives — derivatives w.r.t. complex numbers

$$J_T(\{z_k\}) = J_T(\{\mathrm{Re}[z_k], \mathrm{Im}[z_k]\}); \qquad J_T \in \mathbb{R}, \quad z_k \in \mathbb{C}$$

$$\frac{\partial J_T(\{z_k\})}{\partial \epsilon_{nl}} = \sum_k \left(\frac{\partial J_T}{\partial \mathrm{Re}[z_k]} \frac{\partial \mathrm{Re}[z_k]}{\partial \epsilon_{nl}} + \frac{\partial J_T}{\partial \mathrm{Im}[z_k]} \frac{\partial \mathrm{Im}[z_k]}{\partial \epsilon_{nl}} \right); \qquad \epsilon_{nl} \in \mathbb{R}$$

Define
$$\frac{\partial J_T(\{z_k\})}{\partial z_k} \equiv \frac{1}{2} \left(\frac{\partial J_T}{\partial \mathrm{Re}[z_k]} - i \frac{\partial J_T}{\partial \mathrm{Im}[z_k]} \right)$$
$$\frac{\partial J_T(\{z_k\})}{\partial z_k^*} \equiv \frac{1}{2} \left(\frac{\partial J_T}{\partial \mathrm{Re}[z_k]} + i \frac{\partial J_T}{\partial \mathrm{Im}[z_k]} \right) = \left(\frac{\partial J_T}{\partial z_k} \right)^*$$

$$\frac{\partial J_T(\{z_k\})}{\partial \epsilon_{nl}} = \sum_k \left(\frac{\partial J_T}{\partial z_k} \frac{\partial z_k}{\partial \epsilon_{nl}} + \frac{\partial J_T}{\partial z_k^*} \frac{\partial z_k^*}{\partial \epsilon_{nl}} \right) = 2\mathrm{Re} \left[\sum_k \frac{\partial J_T}{\partial z_k} \frac{\partial z_k}{\partial \epsilon_{nl}} \right]$$

Gradient of Time Evolution Operator

$$
\begin{pmatrix}
\frac{\partial \hat{\mathrm{U}}_n^\dagger}{\partial \epsilon_{n1}} \vert \chi_k(t_n) \rangle \\
\vdots \\
\frac{\partial \hat{\mathrm{U}}_n^\dagger}{\partial \epsilon_{nL}} \vert \chi_k(t_n) \rangle \\
\hat{\mathrm{U}}_n^\dagger \vert \chi_k(t_n) \rangle
\end{pmatrix}
= \exp \left[-\mathrm{i}
\begin{pmatrix}
\hat{\mathrm{H}}_n^\dagger & 0 & \dots & 0 & \hat{\mathrm{H}}_n^{(1)\dagger} \\
0 & \hat{\mathrm{H}}_n^\dagger & \dots & 0 & \hat{\mathrm{H}}_n^{(2)\dagger} \\
\vdots & & \ddots & & \vdots \\
0 & 0 & \dots & \hat{\mathrm{H}}_n^\dagger & \hat{\mathrm{H}}_n^{(L)\dagger} \\
0 & 0 & \dots & 0 & \hat{\mathrm{H}}_n^\dagger\,,
\end{pmatrix}
dt_n \right]
\begin{pmatrix}
0 \\
\vdots \\
0 \\
\vert \chi_k(t_n) \rangle
\end{pmatrix}
$$

$$
\hat{\mathrm{U}}_n = \exp[-\mathrm{i}\hat{\mathrm{H}}_n dt_n]\,; \qquad \hat{\mathrm{H}}_n^{(l)} = \frac{\partial \hat{\mathrm{H}}_n}{\partial \epsilon_l(t)}
$$

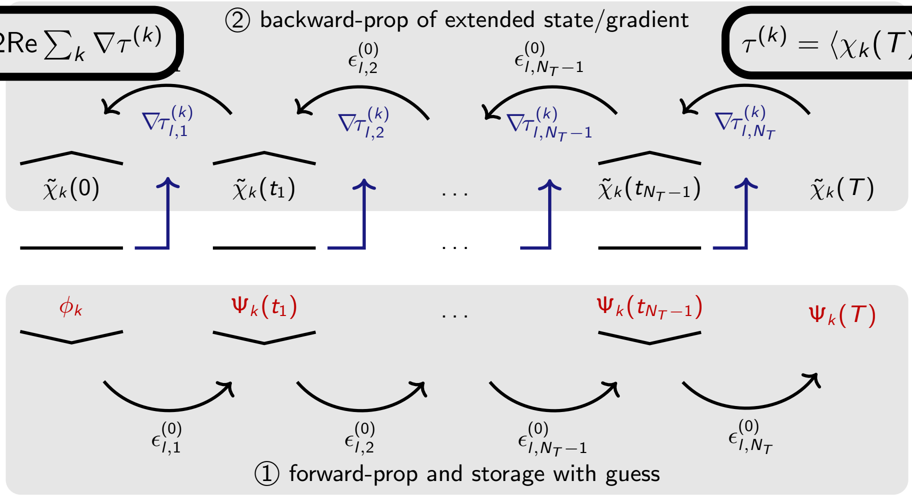
— Goodwin, Kuprov, J. Chem. Phys. 143, 084113 (2015)

`https://github.com/JuliaQuantumControl/QuantumGradientGenerators.jl`

Generalized GRAPE scheme

$$\boxed{\nabla J_T = 2\mathrm{Re} \sum_k \nabla \tau^{(k)}}$$

$$\boxed{\tau^{(k)} = \langle \chi_k(T) \,|\, \Psi_k(T) \rangle}$$



— Goerz *et al.* Quantum 6, 871 (2022)

Semi-Automatic Differentiation

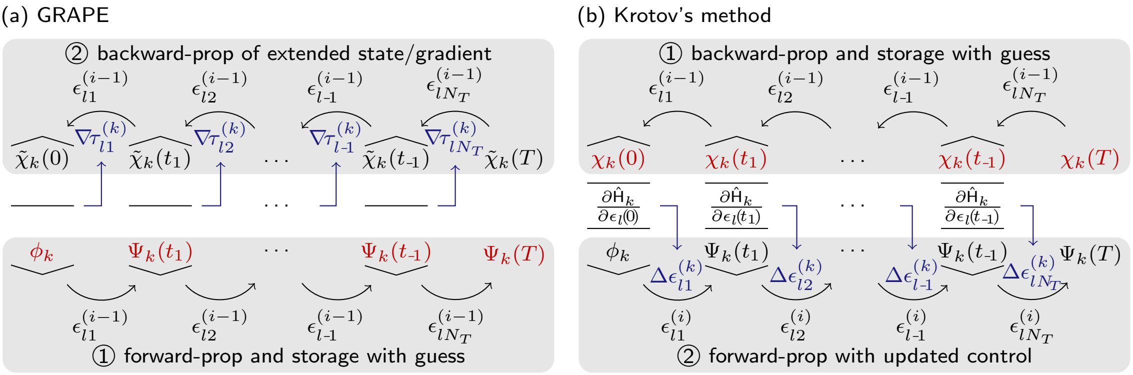
$$|\chi_k(T)\rangle = \frac{\partial J_T}{\partial \langle\Psi_k(T)|}$$

is the only thing evaluated inside AD framework

→ $J_T = J_T(\hat{\mathrm{U}})$

→ $J_T = J_T(\{\tau_k\})$ with $\tau_k = \left\langle \Psi_k(T) \,\middle|\, \Psi_k^{\text{tgt}} \right\rangle$

GRAPE and Krotov Numerical Scheme Comparison



concurrent update

sequential update

— Goerz *et al.* Quantum 6, 871 (2022)

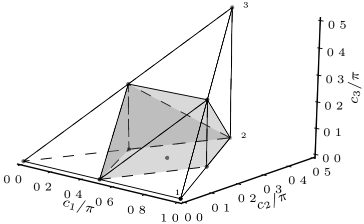
Optimizing for a Maximally Entangling Gate

Cartan decomposition

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

$\hat{\mathrm{k}}_{1,2}$: Single qubit gates; $c_{1,2,3}$: Weyl chamber coordinates

Zhang *et al.* Phys. Rev. A 67, 042313 (2003)



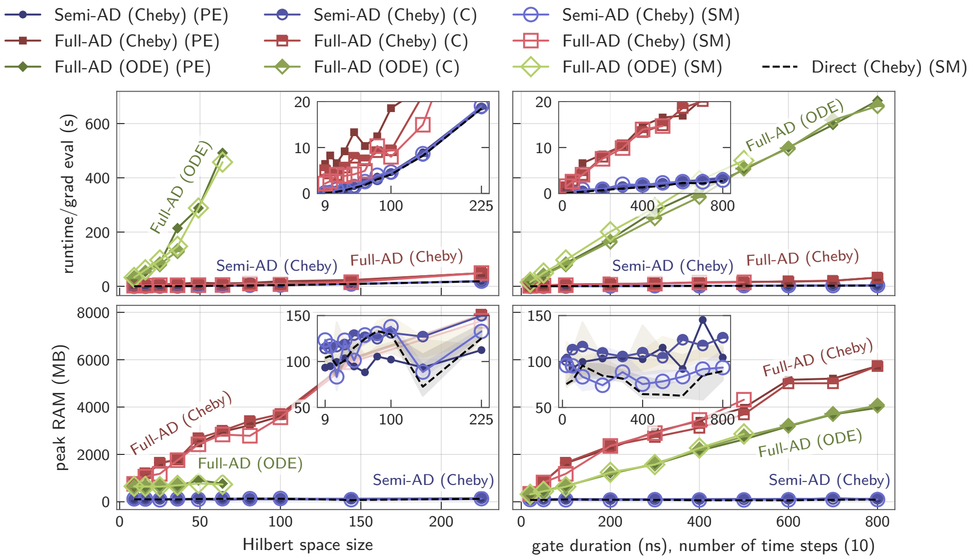
Gate concurrence of two-qubit gate $\hat{\mathrm{U}}$

1. $c_1, c_2, c_3 \propto \operatorname{eigvals}\left(\hat{\mathrm{U}}\tilde{\mathrm{U}}\right)$; $\tilde{\mathrm{U}} = (\hat{\sigma}_y \otimes \hat{\sigma}_y)\, \hat{\mathrm{U}}\, (\hat{\sigma}_y \otimes \hat{\sigma}_y)$
2. $C(\hat{\mathrm{U}}) = \max \left|\sin(c_{1,2,3} \pm c_{3,1,2})\right|$

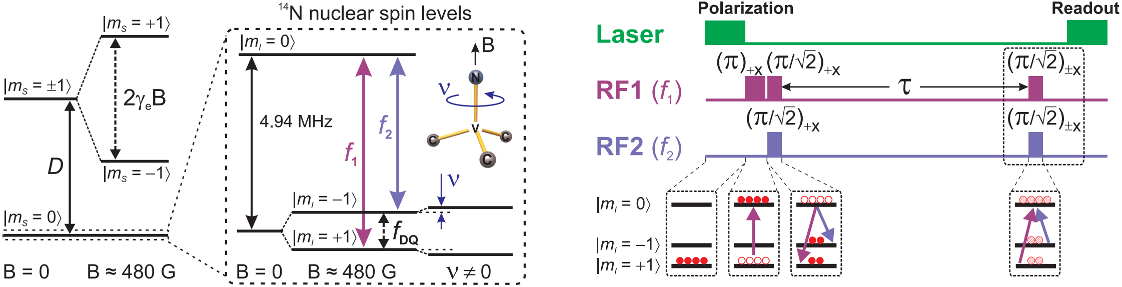
Not analytic!

Childs *et al.* Phys. Rev. A 68, 052311 (2003)

Benchmarks

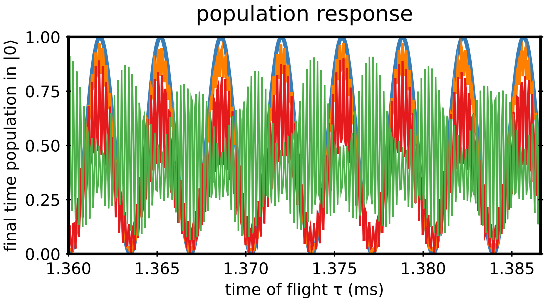

Semi-AD (Cheby) (PE)
Full-AD (Cheby) (PE)
Full-AD (ODE) (PE)
Semi-AD (Cheby) (C)
Full-AD (Cheby) (C)
Full-AD (ODE) (C)
Semi-AD (Cheby) (SM)
Full-AD (Cheby) (SM)
Full-AD (ODE) (SM)
Direct (Cheby) (SM)
runtime/grad eval (s)
peak RAM (MB)
Full-AD (ODE)
Semi-AD (Cheby)
Full-AD (Cheby)
Hilbert space size
gate duration (ns), number of time steps (10)

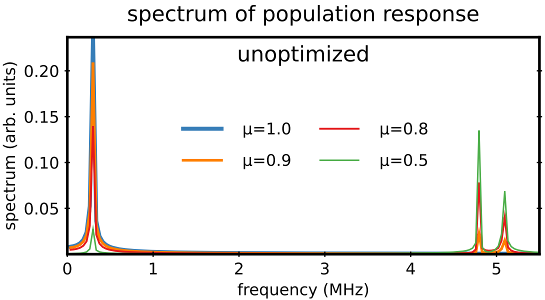
Nuclear Spin Gyroscope

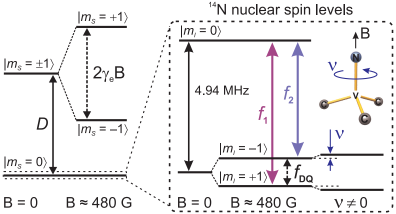


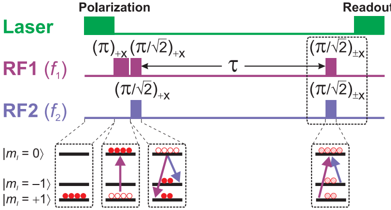
— Adapted from Fig 2 of Jarmola *et. al.* Sci. Adv. 7, eabl3840 (2021)

Optimization of Signal Spectrum

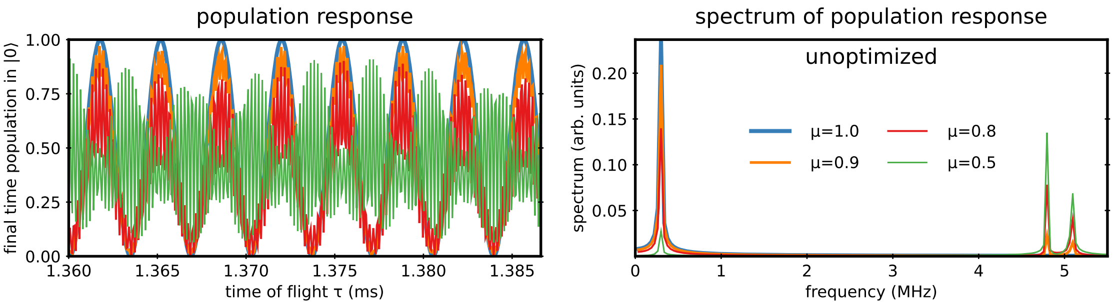








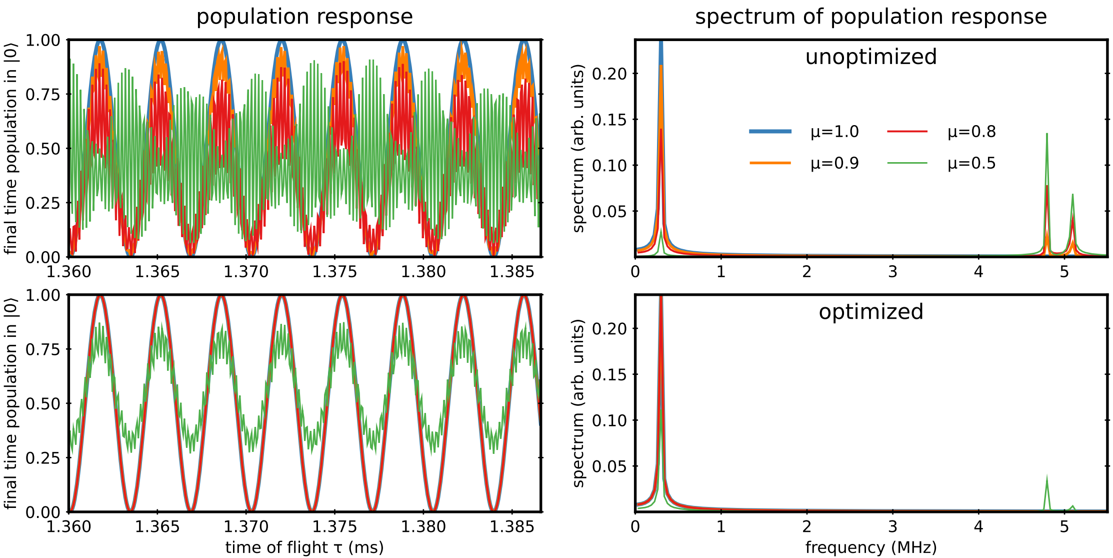
Optimization of Signal Spectrum



$$J_T(\{|\Psi_{\mu,\tau}(T)\rangle\}) = \sum_{\mu} |\mathrm{FFT}([P_0(\tau;\mu)]) - \mathrm{FFT}([P_0(\tau;\mu=1)])|$$

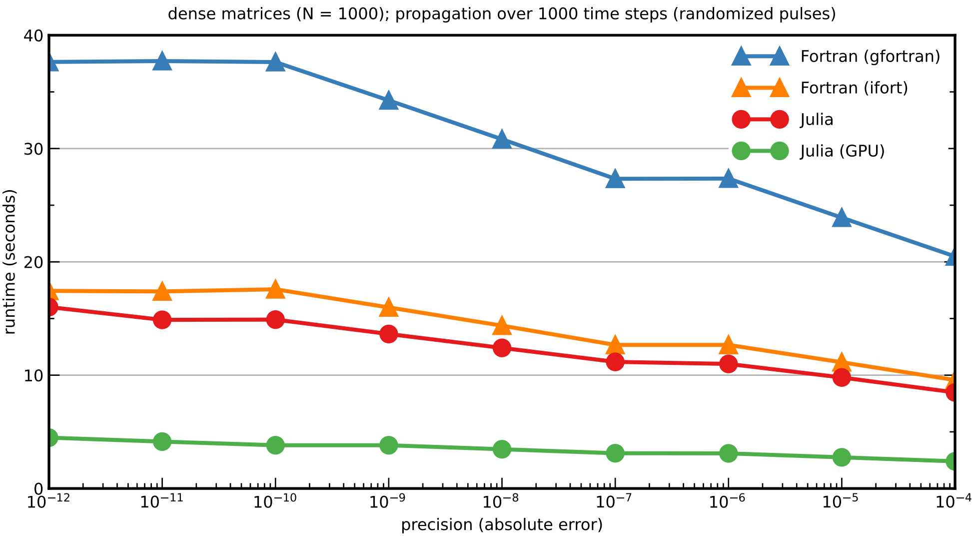
Make spectrum for any μ look like spectrum for $\mu = 1$

Optimization of Signal Spectrum

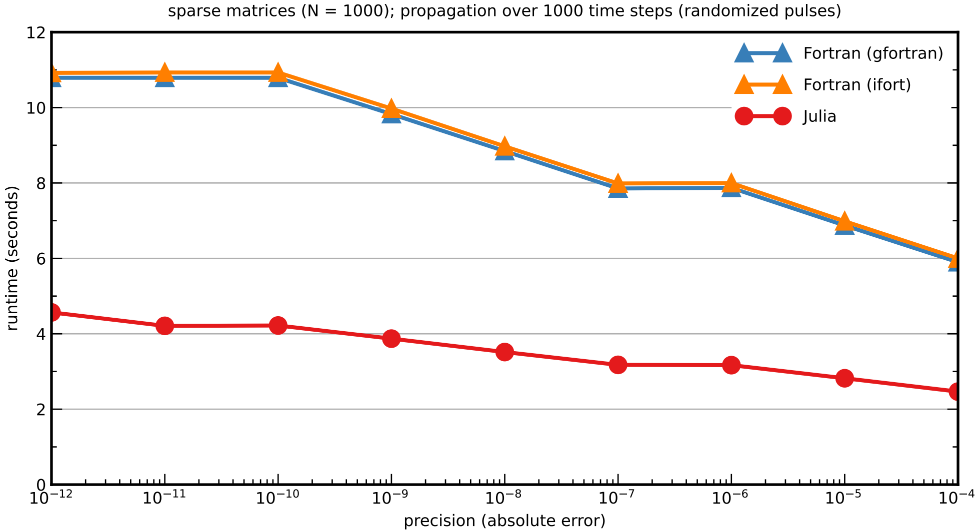


Performance

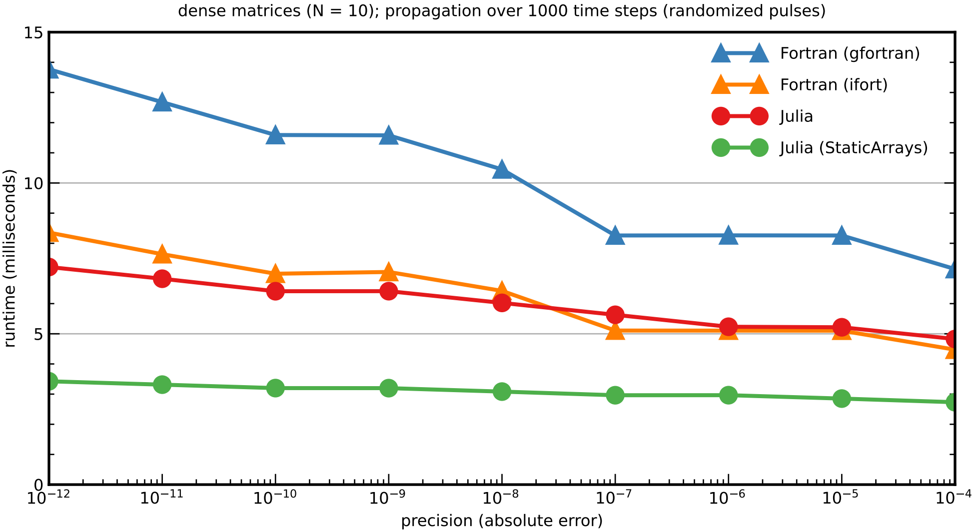
Benchmark for Chebychev Propagator – Large Hilbert Space



Benchmark for Chebychev Propagator – Large Hilbert Space (sparse)



Benchmark for Chebychev Propagator – Small Hilbert Space



Outlook

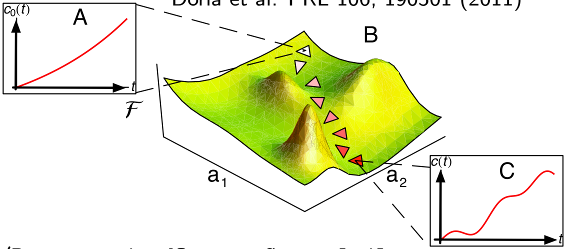
- Parameterized Pulses

$$\epsilon(t) = \epsilon(\{u_n\}, t)$$

  - experimental constraints
  - no PWC error
  - but: local traps, controllability issues

  `https://github.com/JuliaQuantumControl/ParameterizedQuantumControl.jl`
  (CRAB, GOAT, GROUP, ...)

Doria et al. PRL 106, 190501 (2011)



- Semi-Classical Optimization
- Reinforcement Learning
- ...

Gradients of parametrized pulses

$$
\begin{pmatrix}
\frac{\partial \hat{\mathrm{U}}}{\partial u_1} \vert \Psi_k \rangle \\
\vdots \\
\frac{\partial \hat{\mathrm{U}}}{\partial u_N} \vert \Psi_k \rangle \\
\hat{\mathrm{U}} \vert \Psi_k \rangle
\end{pmatrix}
= \exp \left[-\mathrm{i} \mathcal{T} \int_0^T
\begin{pmatrix}
\hat{\mathrm{H}}(t) & 0 & \dots & 0 & \hat{\mathrm{H}}^{(1)}(t) \\
0 & \hat{\mathrm{H}}(t) & \dots & 0 & \hat{\mathrm{H}}^{(2)}(t) \\
\vdots & & \ddots & & \vdots \\
0 & 0 & \dots & \hat{\mathrm{H}}(t) & \hat{\mathrm{H}}^{(N)}(t) \\
0 & 0 & \dots & 0 & \hat{\mathrm{H}}(t)
\end{pmatrix} dt \right]
\begin{pmatrix}
0 \\
\vdots \\
0 \\
\vert \Psi_k \rangle
\end{pmatrix}
$$

with $\hat{\mathrm{H}}^{(n)}(t) = \frac{\partial \hat{\mathrm{H}}(t)}{\partial u_n}$

— "GOAT": Machnes *et al.* Phys. Rev. Lett. 120, 150401 (2018)

`https://github.com/JuliaQuantumControl/QuantumGradientGenerators.jl`

Conclusion

- Julia: multiple dispatch for flexibility and performance
- QuantumControl framework: general structure of optimal control
- Rotating Tractor Atom Interferometer: project-specific data structures
- Semi-automatic differentiation
- Nuclear Spin Gyroscope: optimize spectrum of response
- Performance: Julia matches or outperforms Fortran

Thank You!

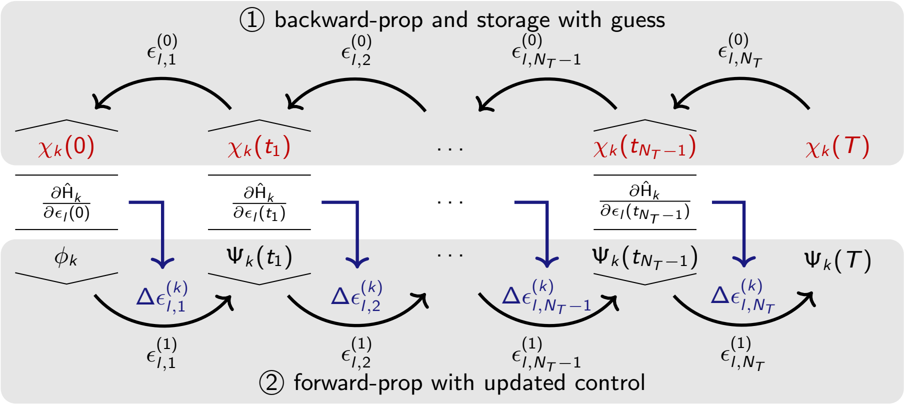
Krotov's Method

$$J(\epsilon(t)) = J_T(\{|\Psi_k(T)\rangle\}) + \lambda_a \int_0^T \frac{(\Delta\epsilon(t))^2}{S(t)} dt$$

$$\Downarrow$$

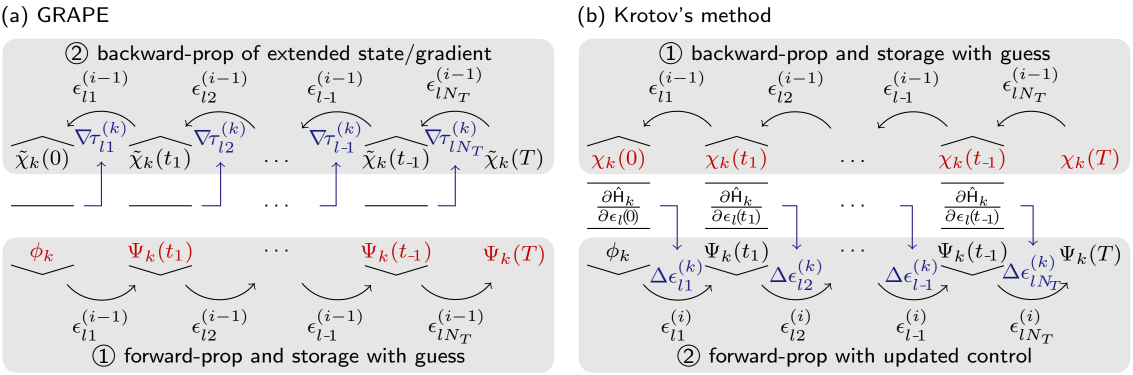
$$\Delta\epsilon(t) = \frac{S(t)}{\lambda_a} \left\langle \chi_k^{(0)}(t) \middle| \frac{\partial H}{\partial \epsilon(t)} \middle| \Psi_k^{(1)}(t) \right\rangle$$

Krotov Numerical Scheme



— Goerz *et al.* Quantum 6, 871 (2022)

GRAPE and Krotov Numerical Scheme Comparison



concurrent update

sequential update

— Goerz *et al.* Quantum 6, 871 (2022)

Open Quantum Systems

Lindblad equation:

$$
\begin{aligned}
\frac{d}{dt}\hat{\rho}(t) &= -i\left[\hat{\mathsf{H}}, \hat{\rho}(t)\right] + \mathcal{L}_D(\hat{\rho}(t)) \\
&= -i\left[\hat{\mathsf{H}}, \hat{\rho}(t)\right] + \sum_k \left(\hat{\mathsf{A}}_k \hat{\rho} \hat{\mathsf{A}}_k^\dagger - \frac{1}{2}\hat{\mathsf{A}}_k^\dagger \hat{\mathsf{A}}_k \hat{\rho} - \frac{1}{2}\hat{\rho}\hat{\mathsf{A}}_k^\dagger \hat{\mathsf{A}}_k\right)
\end{aligned}
$$

Vectorization rule:

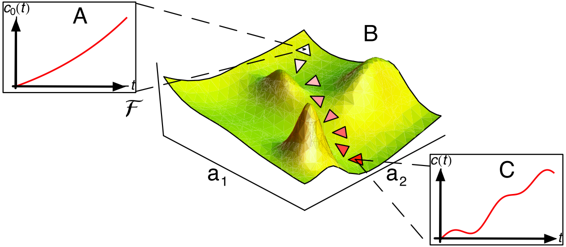
$$
\operatorname{vec}\left(\hat{\mathsf{A}}\hat{\rho}\hat{\mathsf{B}}\right) = \left(\hat{B}^T \otimes \hat{A}\right)\vec{\rho}
$$

Matrix representation of Lindbladian:

$$
\hat{L} = -i(\mathbf{1} \otimes \hat{H}) + i(\hat{H}^T \otimes \mathbf{1}) + \sum_k \left[(\hat{A}_k^\dagger)^T \otimes \hat{A}_k - \frac{1}{2}\left(\mathbf{1} \otimes \hat{A}_k^\dagger \hat{A}_k\right) - \frac{1}{2}\left((\hat{A}_k^\dagger \hat{A}_k)^T \otimes \mathbf{1}\right)\right]
$$

— Goerz *et. al.* arXiv:1312.0111v2 (2021), Appendix B

Gradient-free optimization



Doria et al. PRL 106, 190501 (2011)

e.g. Nelder-Mead (simplex), genetic algorithms. . .