QuantumControl.jl: A modern framework for quantum optimal control

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

DEVCOM Army Research Lab

APS March Meeting 2023

JuliaQuantumControl

github.com/JuliaQuantumControl

JuliaQuantumControl

Julia Framework for Quantum Optimal Control

12 followers · https://juliaquantumcontrol.github.i...

Unfollow

Overview · Repositories 15 · Discussions · Projects · Packages · Teams 1 · People 4 · Settings

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.

View as: **Public**

You are viewing the README and pinned repositories as a public user.

Get started with tasks that most successful organizations complete.

Top discussions this past month

Discussions are for sharing announcements, creating conversation in your community, answering questions, and more.

Start a new discussion

JuliaQuantumControl

github.com/JuliaQuantumControl

Package	Version	CI Status	Coverage	Description
⭐ QuantumPropagators.jl	Mar 2023 v0.4.2	CI passing	codecov 84%	Simulate the time evolution of quantum systems (docs)
QuantumControlBase.jl	Mar 2023 v0.8.1	CI passing	codecov 88%	Shared methods and data structures (docs)
QuantumGradientGenerators.jl	Feb 2023 v0.1.1	CI passing	codecov 79%	Dynamic Gradients for Quantum Control (docs)
Krotov.jl	Mar 2023 v0.5.2	CI passing	codecov 90%	Krotov's method of optimal control (docs)
GRAPE.jl	Mar 2023 v0.5.3	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)
⭐ QuantumCitations.jl	Mar 2023 v0.2.1	CI passing	codecov 90%	Documenter plugin for BibTeX citations and references (docs)
QuantumControlTestUtils.jl	Mar 2023 v0.1.3	CI passing		Tools for testing and benchmarking (docs)
⭐ QuantumControl.jl	Mar 2023 v0.6.2	CI passing	codecov 76%	Framework for Quantum Dynamics and Control (docs)

Top languages

Julia Makefile

Most used topics Manage

julia quantum grape optimal-control numerical-methods

Julia

julialang.org

Julia in a Nutshell

Fast

Julia was designed from the beginning for high performance. Julia programs compile to efficient native code for multiple platforms via LLVM.

Dynamic

Julia is dynamically typed, feels like a scripting language, and has good support for interactive use.

Reproducible

Reproducible environments make it possible to recreate the same Julia environment every time, across platforms, with pre-built binaries.

Composable

Julia uses multiple dispatch as a paradigm, making it easy to express many object-oriented and functional programming patterns. The talk on the Unreasonable Effectiveness of Multiple Dispatch explains why it works so well.

General

Julia provides asynchronous I/O, metaprogramming, debugging, logging, profiling, a package manager, and more. One can build entire Applications and Microservices in Julia.

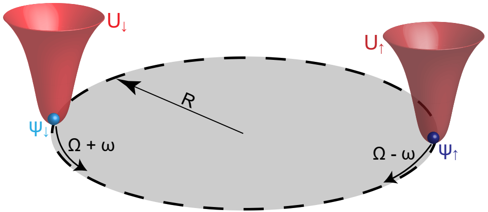
Open source

Julia is an open source project with over 1,000 contributors. It is made available under the MIT license. The source code is available on GitHub.

See Julia Code Examples | Try Julia In Your Browser

Flexibility

Rotating Tractor Interferometer



B. Dash *et al.* "Rotation sensing using tractor atom interferometry" (in preparation)

$$\hat{H}_{\pm} = -\frac{\hbar^2}{2mR^2}\frac{\partial^2}{\partial\theta^2} + V_0 \cos\left[m(\theta + \phi_{\pm}(t))\right]$$

typically: $\hat{H} = \hat{H}_0 + \epsilon(t)\hat{H}_1$

with control $\epsilon(t)$

here: $\hat{H} = \hat{T} + \hat{V}(\theta \pm \phi(t))$

with control $\phi(t)$

Multiple Dispatch for $\hat{\mathrm{H}} = \hat{\mathrm{T}} + \hat{\mathrm{V}}(\theta \pm \phi(t))$

Default

```
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
```

include/rotating_tai.jl

```
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% ¶ 38/434: 1⟩ "α include/rotating_tai.jl ⟨julia⟨master

Arbitrary Functionals

Quantum Gate Concurrence: Max concurrence of $\hat{\mathrm{U}} \left|\Psi\right\rangle$ for separable input state $\left|\Psi\right\rangle$

Given two-qubit gate $\hat{\mathrm{U}}$ with $U_{ij} = \langle \Phi_i \,|\, \Psi_j(T) \rangle$ for $\left|\phi_i\right\rangle = \left|00\right\rangle, \left|01\right\rangle, \left|10\right\rangle, \left|11\right\rangle$

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

$$\Rightarrow \quad J_T(\hat{\mathrm{U}}) = \frac{1}{2}\left(1 - C(\hat{\mathrm{U}})\right) + \frac{1}{2}\left(1 - \underbrace{\frac{1}{4}\mathrm{tr}[\hat{\mathrm{U}}\hat{\mathrm{U}}^\dagger]}_{\text{unitarity}}\right)$$

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

Not analytic!

Arbitrary Functionals

Quantum Gate Concurrence: Max concurrence of $\hat{\mathrm{U}}\,|\Psi\rangle$ for separable input state $|\Psi\rangle$

Given two-qubit gate $\hat{\mathrm{U}}$ with $U_{ij} = \langle \Phi_i \,|\, \Psi_j(T) \rangle$ for $|\phi_i\rangle = |00\rangle\,, |01\rangle\,, |10\rangle\,, |11\rangle$

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

$$\Rightarrow \quad J_T(\hat{\mathrm{U}}) = \frac{1}{2}\Big(1 - C(\hat{\mathrm{U}})\Big) + \frac{1}{2}\Big(1 - \underbrace{\frac{1}{4}\mathrm{tr}[\hat{\mathrm{U}}\hat{\mathrm{U}}^\dagger]}_{\text{unitarity}}\Big)$$

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

Semi-automatic differentiation: Calculate $\frac{\partial J_T}{\partial \langle \Psi_k(T)|}$ via automatic differentiation.

$\Rightarrow$ automatic gradients for arbitrary functionals with no numerical overhead compared to analytical gradients

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

Example: Gate Concurrence Maximization

localhost:61816/notebooks/perfect_entanglers.ipynb

File Edit View Insert Cell Kernel Navigate Widgets Help Trusted Julia 1.8 (auto threads)

```julia

tlist, Ωre_guess, Ωim_guess = guess_amplitudes();
```

In [8]:

```julia
function transmon_hamiltonian(;
    Ωre, Ωim, N=N, ω₁=4.380GHz, ω₂=4.614GHz, ωd=4.498GHz, α₁=-210MHz,
    α₂=-215MHz, J=-3MHz, λ=1.03,
)
    𝟙 = SparseMatrixCSC{ComplexF64,Int64}(sparse(I, N, N))
    b̂₁ = spdiagm(1 => complex.(sqrt.(collect(1:N-1)))) ⊗ 𝟙
    b̂₂ = 𝟙 ⊗ spdiagm(1 => complex.(sqrt.(collect(1:N-1))))
    b̂₁⁺ = sparse(b̂₁'); b̂₂⁺ = sparse(b̂₂')
    n̂₁ = sparse(b̂₁' * b̂₁); n̂₂ = sparse(b̂₂' * b̂₂)
    n̂₁² = sparse(n̂₁ * n̂₁); n̂₂² = sparse(n̂₂ * n̂₂)
    b̂₁⁺_b̂₂ = sparse(b̂₁' * b̂₂); b̂₁_b̂₂⁺ = sparse(b̂₁ * b̂₂')

    ω̃₁ = ω₁ - ωd; ω̃₂ = ω₂ - ωd

    Ĥ₀ = sparse(
        (ω̃₁ - α₁ / 2) * n̂₁ +
        (α₁ / 2) * n̂₁² +
        (ω̃₂ - α₂ / 2) * n̂₂ +
        (α₂ / 2) * n̂₂² +
        J * (b̂₁⁺_b̂₂ + b̂₁_b̂₂⁺)
    )
    Ĥ₁re = sparse((1 / 2) * (b̂₁ + b̂₁⁺ + λ * b̂₂ + λ * b̂₂⁺))
    Ĥ₁im = sparse((𝕚 / 2) * (b̂₁⁺ - b̂₁ + λ * b̂₂⁺ - λ * b̂₂))
    return hamiltonian(Ĥ₀, (Ĥ₁re, Ωre), (Ĥ₁im, Ωim))
end;
```

In [9]:

```julia
H = transmon_hamiltonian(Ωre=Ωre_guess, Ωim=Ωim_guess);
```

Example: Gate Concurrence Maximization

localhost:61816/notebooks/perfect_entanglers.ipynb

File Edit View Insert Cell Kernel Navigate Widgets Help Trusted Julia 1.8 (auto threads)

In [9]:
```julia
H = transmon_hamiltonian(Ωre=Ωre_guess, Ωim=Ωim_guess);
```

1.2 Maximization of Gate Concurrence

In [10]:
```julia
J_T_C = U -> 0.5 * (1 - gate_concurrence(U)) + 0.5 * (1 - unitarity(U));
```

In [11]:
```julia
basis = [ket("00"), ket("01"), ket("10"), ket("11")];
```

In [12]:
```julia
objectives = [Objective(; initial_state=Ψ, generator=H) for Ψ ∈ basis];
```

In [13]:
```julia
problem = ControlProblem(
    objectives=objectives,
    tlist=tlist,
    iter_stop=100,
    J_T=gate_functional(J_T_C),
    chi=make_gate_chi(J_T_C, objectives),
    check_convergence=res -> begin
        (
            (res.J_T <= 1e-3) &&
            (res.converged = true) &&
            (res.message = "Found a perfect entangler")
        )
    end,
    use_threads=true,
);
```

In [15]:
```julia
res = optimize(problem; method=:GRAPE)
```

Example: Gate Concurrence Maximization

localhost:61816/notebooks/perfect_entanglers.ipynb

File Edit View Insert Cell Kernel Navigate Widgets Help Trusted Julia 1.8 (auto threads)

In [15]:
```
res = optimize(problem; method=:GRAPE)
```

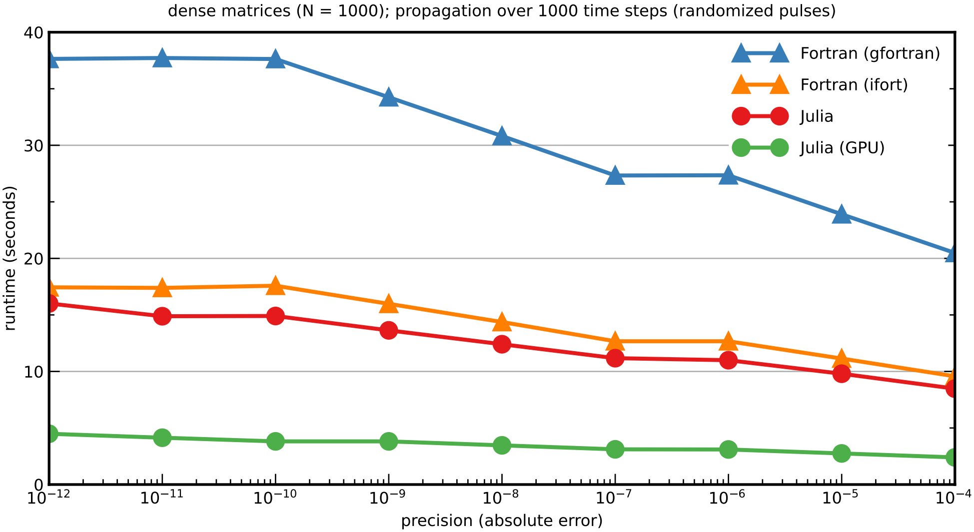
```
 iter.        J_T     |∇J_T|       ΔJ_T   FG(F)    secs
     0   1.57e-01   1.42e-01        n/a    1(0)     0.5
     1   1.46e-01   3.18e-01  -1.05e-02    1(0)     0.2
     2   1.30e-01   2.86e-01  -1.61e-02    1(0)     0.2
     3   8.10e-02   2.10e-01  -4.91e-02    2(0)     0.4
     4   7.66e-02   3.79e-01  -4.41e-03    1(0)     0.3
     5   4.89e-02   1.87e-01  -2.77e-02    1(0)     0.2
     6   2.64e-02   2.11e-01  -2.25e-02    1(0)     0.2
     7   7.54e-03   1.09e-01  -1.89e-02    1(0)     0.2
     8   5.86e-03   1.98e-01  -1.68e-03    1(0)     0.2
     9   3.00e-03   4.01e-02  -2.87e-03    1(0)     0.2
    10   2.71e-03   2.72e-02  -2.88e-04    1(0)     0.2
    11   2.21e-03   2.82e-02  -5.01e-04    1(0)     0.3
    12   1.42e-03   2.46e-02  -7.84e-04    1(0)     0.2
    13   3.24e-04   2.83e-02  -1.10e-03    1(0)     0.2
```

Out[15]:
```
GRAPE Optimization Result
-------------------------
- Started at 2023-03-20T06:30:20.133
- Number of objectives: 4
- Number of iterations: 13
- Number of pure func evals: 0
- Number of func/grad evals: 15
- Value of functional: 3.24322e-04
- Reason for termination: Found a perfect entangler
- Ended at 2023-03-20T06:30:23.979 (3 seconds, 846 milliseconds)
```

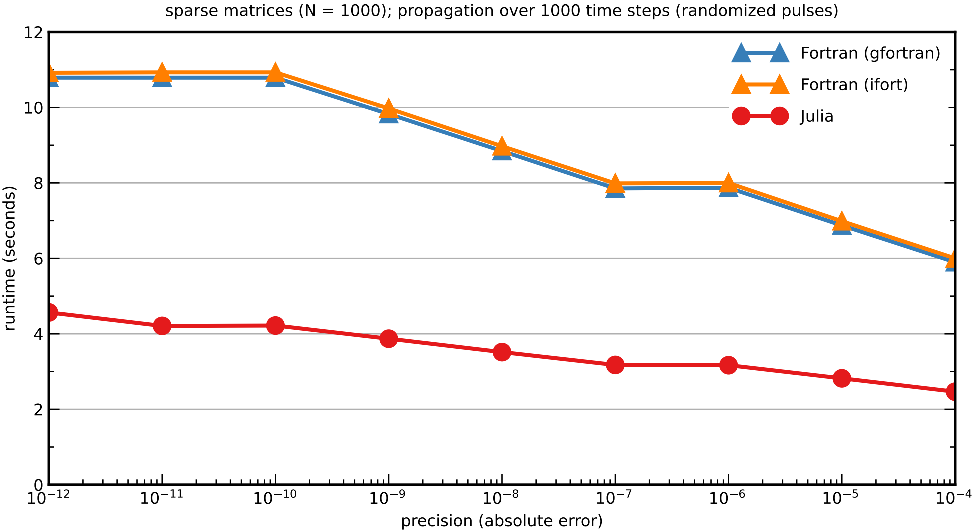
In [16]:
```
ϵ_opt = res.optimized_controls[1] + 𝕚 * res.optimized_controls[2]
Ω_opt = ϵ_opt .* discretize(Ωre_guess.shape, tlist)
```

Performance

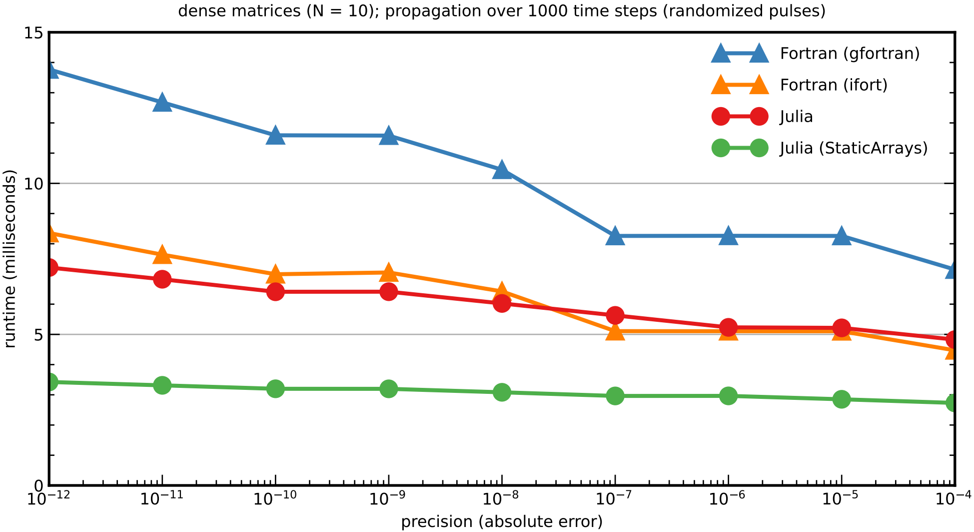
Benchmark for Chebychev Propagator – Large Hilbert Space



Benchmark for Chebychev Propagator – Large Hilbert Space (sparse)



Benchmark for Chebychev Propagator – Small Hilbert Space



Conclusions

`https://github.com/JuliaQuantumControl`

Flexibility:

- Interactive usage (notebooks)
- Use custom project-specific data structures
- Tie into Julia ecosystem (e.g., automatic differentiation, GPU computing)

Performance:

- Out of the Box: match Fortran (ifort + MKL)
- GPU, Sparse Matrices, StaticArrays: beat Fortran ($> 2\times$)

Outlook

https://github.com/JuliaQuantumControl

QuantumPropagators.jl

- Support for time-continuous controls (via DifferentialEquations.jl)

QuantumControl.jl

- Optimization methods for analytical pulse shapes (CRAB, GOAT, ...)
- Reinforcement learning

Users and Contributors welcome!

Please reach out by Email, GitHub, or #quantumcontrol channel on the Julia Slack