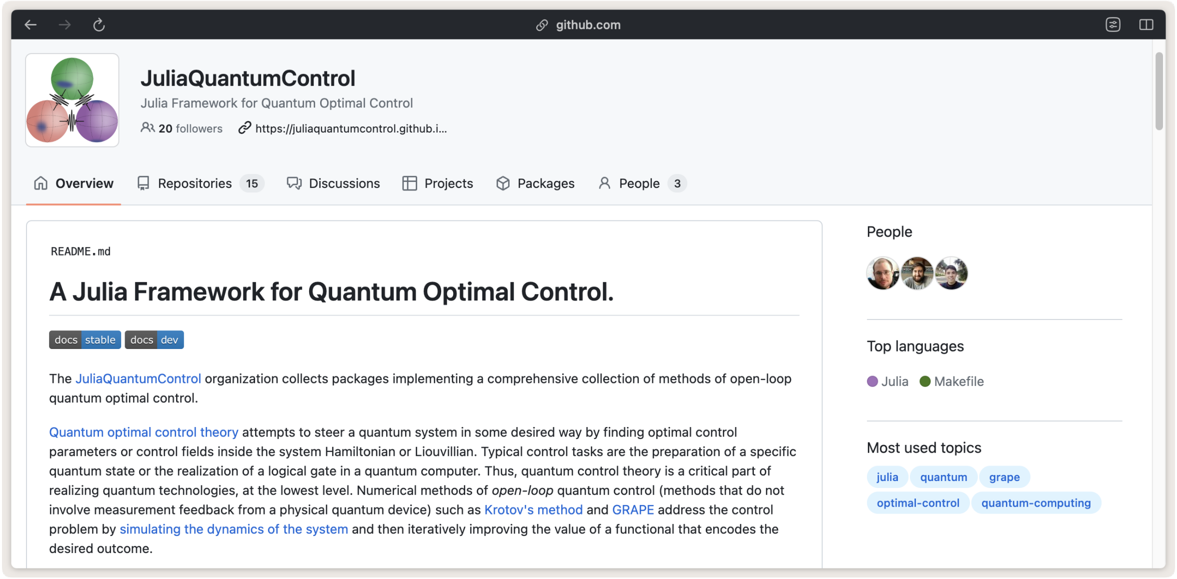
Quantum Dynamics and Control with QuantumControl.jl

Michael H. Goerz

DEVCOM Army Research Lab

JuliaCon 2023

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

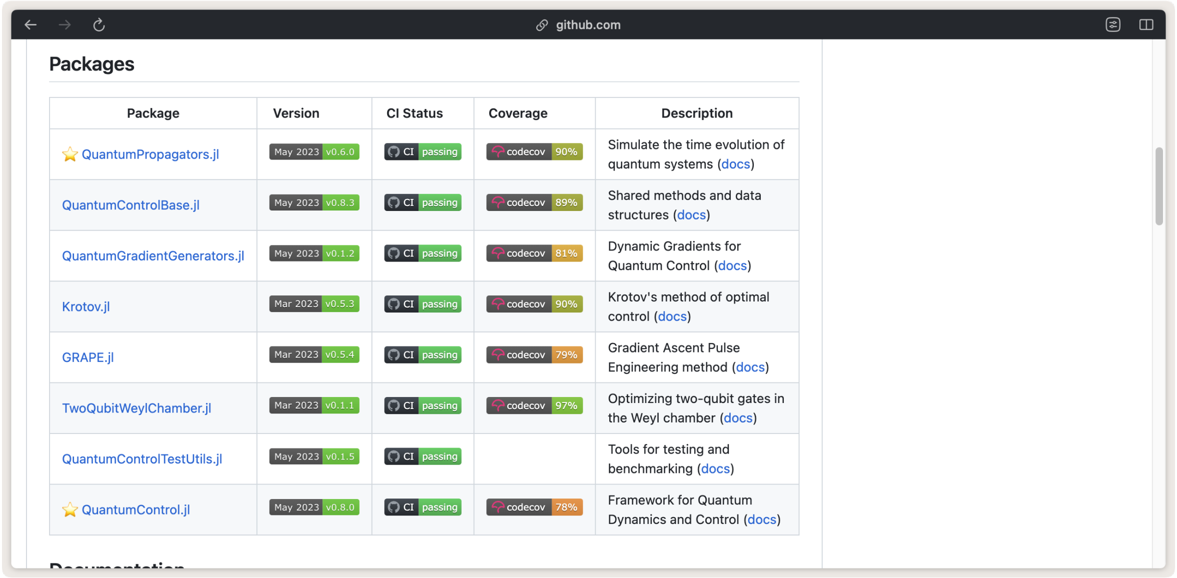
Top languages

● Julia ● Makefile

Most used topics

julia quantum grape optimal-control quantum-computing

JuliaQuantumControl



github.com

Packages

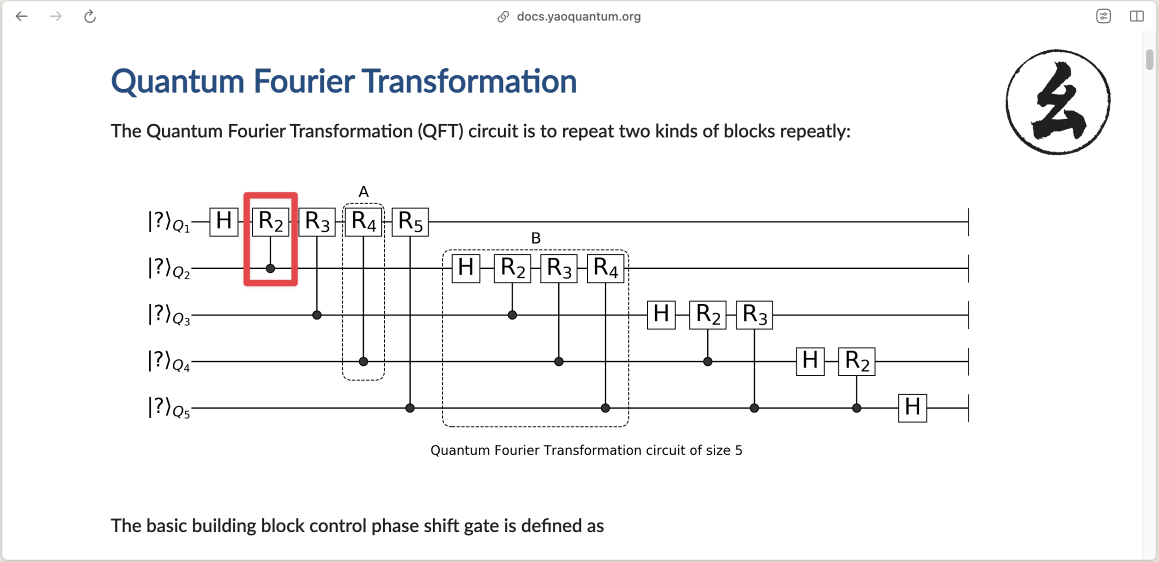
Package	Version	CI Status	Coverage	Description
⭐ QuantumPropagators.jl	May 2023 v0.6.0	CI passing	codecov 90%	Simulate the time evolution of quantum systems (docs)
QuantumControlBase.jl	May 2023 v0.8.3	CI passing	codecov 89%	Shared methods and data structures (docs)
QuantumGradientGenerators.jl	May 2023 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)

UNCLASSIFIED

What is Quantum Control?

Steer a quantum system in some desired way

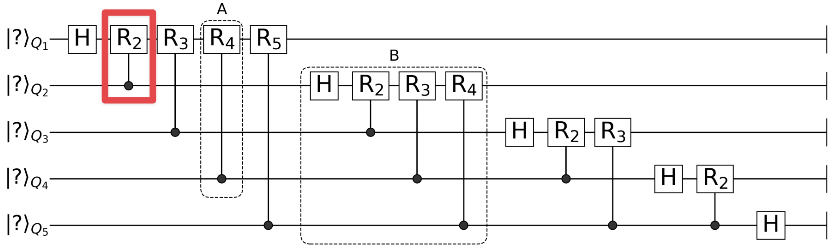
Quantum Gates



docs.yaoquantum.org

Quantum Fourier Transformation

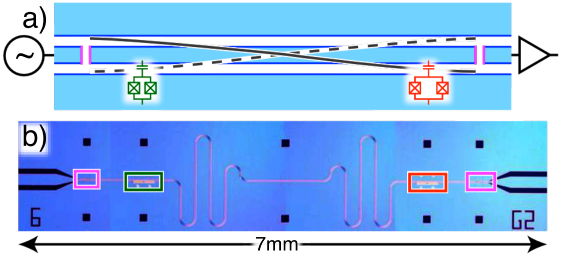
The Quantum Fourier Transformation (QFT) circuit is to repeat two kinds of blocks repeatly:



Quantum Fourier Transformation circuit of size 5

The basic building block control phase shift gate is defined as

Two-Transmon Gate

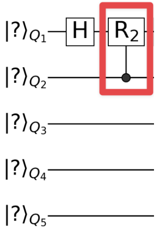


Majer *et al.* Nature 449, 443 (2007)

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

$\uparrow$

microwave field in transmission line

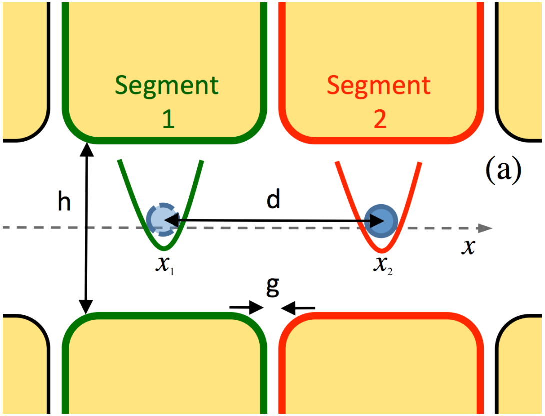


$$\left.\begin{aligned} |00\rangle &\rightarrow \text{CR}_2\,|00\rangle \\ |01\rangle &\rightarrow \text{CR}_2\,|01\rangle \\ |10\rangle &\rightarrow \text{CR}_2\,|10\rangle \\ |11\rangle &\rightarrow \text{CR}_2\,|11\rangle \end{aligned}\right\}$$

with the same $\epsilon(t)$;

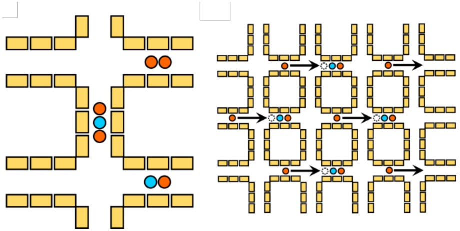
acting on logical subspace

Controlling the transport of an ion



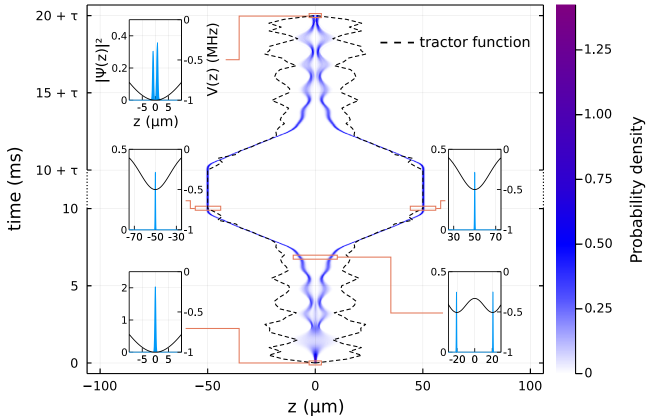
Fürst *et al.* New J. Phys. 16, 075007 (2014)

Find electrode voltages
to move trapped ions



Bruzewicz *et al.* npj Quantum Inf 5, 102 (2019)

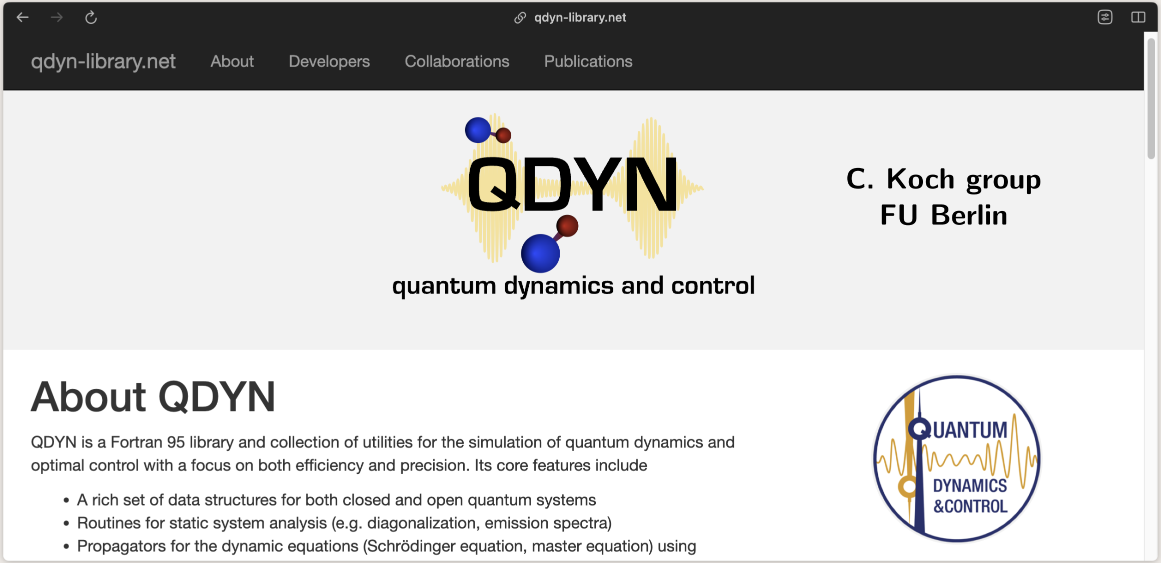
Tractor atom interferometry



Raithel *et al.* Quantum Sci. Technol. 8, 014001 (2022)

Find non-adiabatic tractor potential closing interferometric path

Fortran: QDYN library



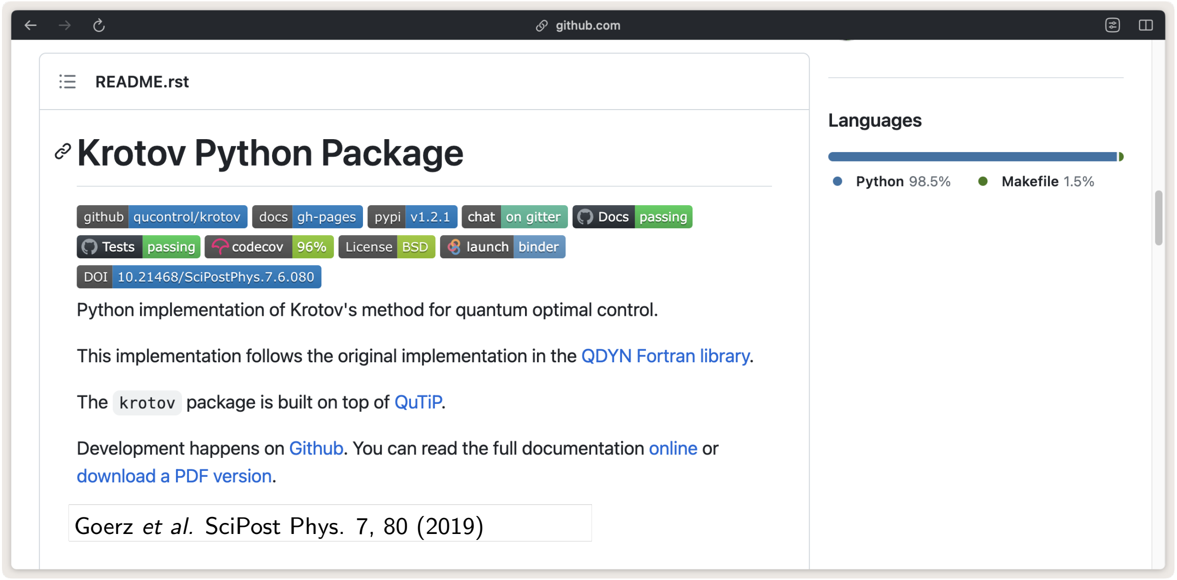
C. Koch group
FU Berlin

About QDYN

QDYN is a Fortran 95 library and collection of utilities for the simulation of quantum dynamics and optimal control with a focus on both efficiency and precision. Its core features include

- A rich set of data structures for both closed and open quantum systems
- Routines for static system analysis (e.g. diagonalization, emission spectra)
- Propagators for the dynamic equations (Schrödinger equation, master equation) using

Python



github.com

README.rst

Krotov Python Package

Python implementation of Krotov's method for quantum optimal control.

This implementation follows the original implementation in the QDYN Fortran library.

The `krotov` package is built on top of QuTiP.

Development happens on Github. You can read the full documentation online or download a PDF version.

Languages

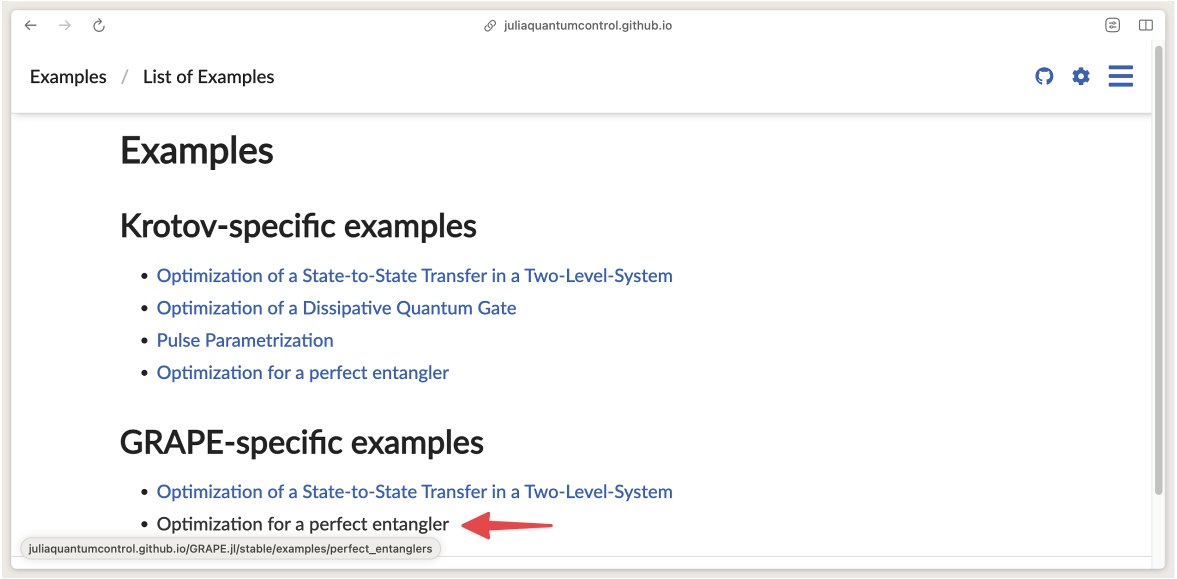
● Python 98.5% ● Makefile 1.5%

Goerz *et al.* SciPost Phys. 7, 80 (2019)

Why Julia?

- Flexibility
- Performance
- Expressiveness

QuantumControl.jl examples



juliaquantumcontrol.github.io

Examples / List of Examples

Examples

Krotov-specific examples

- Optimization of a State-to-State Transfer in a Two-Level-System
- Optimization of a Dissipative Quantum Gate
- Pulse Parametrization
- Optimization for a perfect entangler

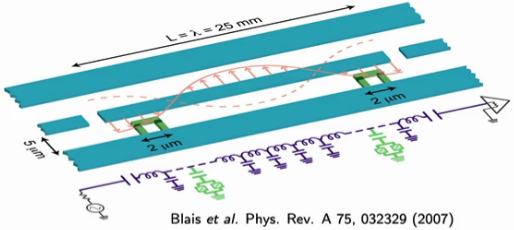
GRAPE-specific examples

- Optimization of a State-to-State Transfer in a Two-Level-System
- Optimization for a perfect entangler

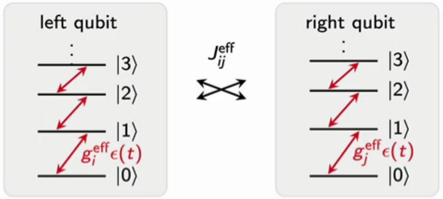
juliaquantumcontrol.github.io/GRAPE.jl/stable/examples/perfect_entanglers

Example: Optimization of Perfectly Entangling Quantum gate

Two Transmon qubits with a shared transmission line ¶



Blais *et al.* Phys. Rev. A 75, 032329 (2007)



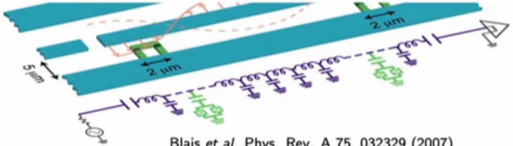
Goerz *et al.* EPJ Quantum Tech. 2, 21 (2015)
Goerz *et al.* npj Quantum Information 3, 37 (2017)

Hamiltonian

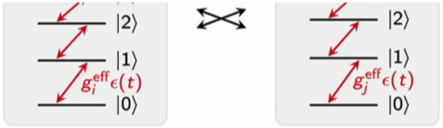
The energies system energies are on the order of GHz (angular frequency; the factor 2π is implicit), with dynamics on the order of ns

```
[ ]: const GHz = 2π
```

UNCLASSIFIED



Blais *et al.* Phys. Rev. A 75, 032329 (2007)



Goerz *et al.* EPJ Quantum Tech. 2, 21 (2015)
Goerz *et al.* npj Quantum Information 3, 37 (2017)

Hamiltonian

The energies system energies are on the order of GHz (angular frequency; the factor 2π is implicit), with dynamics on the order of ns

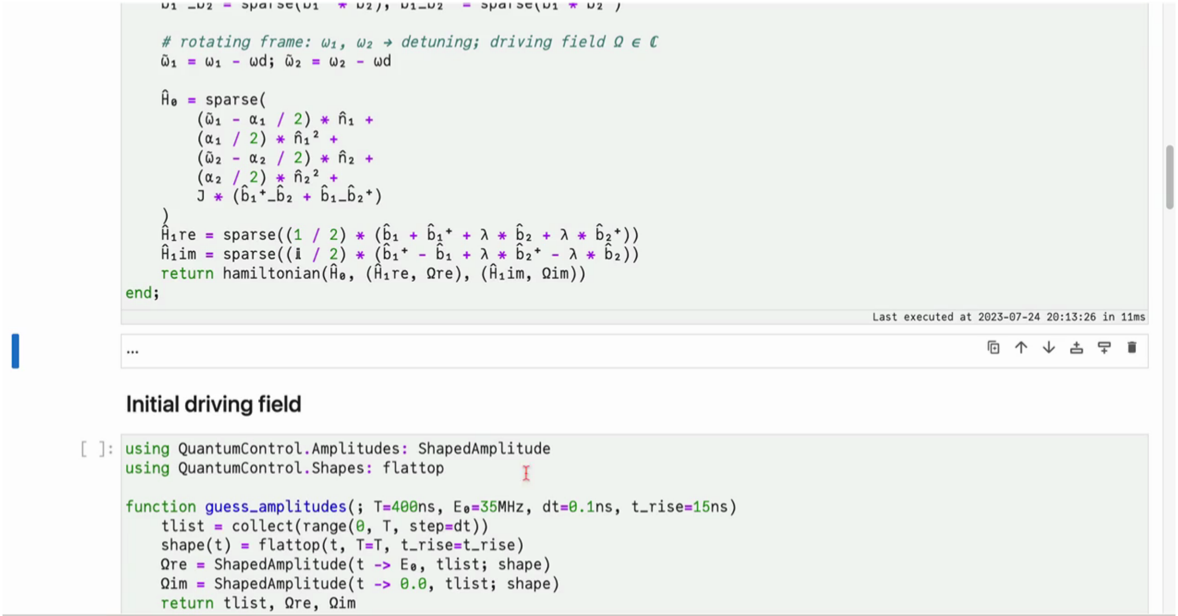
```
[ ]: const GHz = 2π
     const MHz = 0.001GHz
     const ns = 1.0
     const μs = 1000ns;
     ⊗ = kron;
     const 𝕚 = 1im;
```

We truncated the Hamiltonian to N levels

```
[ ]: const N = 6;  # levels per transmon
```

So the dimension of the total Hilbert space is $N^2 = 36$

The Hamiltonian and parameters are taken from Goerz *et al.*, Phys. Rev. A 91, 062307 (2015); Table 1.



```
    b̂₁⁺_b̂₂ = sparse(b̂₁⁺ * b̂₂); b̂₁_b̂₂⁺ = sparse(b̂₁ * b̂₂⁺)

    # rotating frame: ω₁, ω₂ → detuning; driving field Ω ∈ ℂ
    ω̃₁ = ω₁ - ωd; ω̃₂ = ω₂ - ωd

    Ĥ₀ = sparse(
        (ω̃₁ - α₁ / 2) * n̂₁ +
        (α₁ / 2) * n̂₁^2 +
        (ω̃₂ - α₂ / 2) * n̂₂ +
        (α₂ / 2) * n̂₂^2 +
        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;
```

Last executed at 2023-07-24 20:13:26 in 11ms

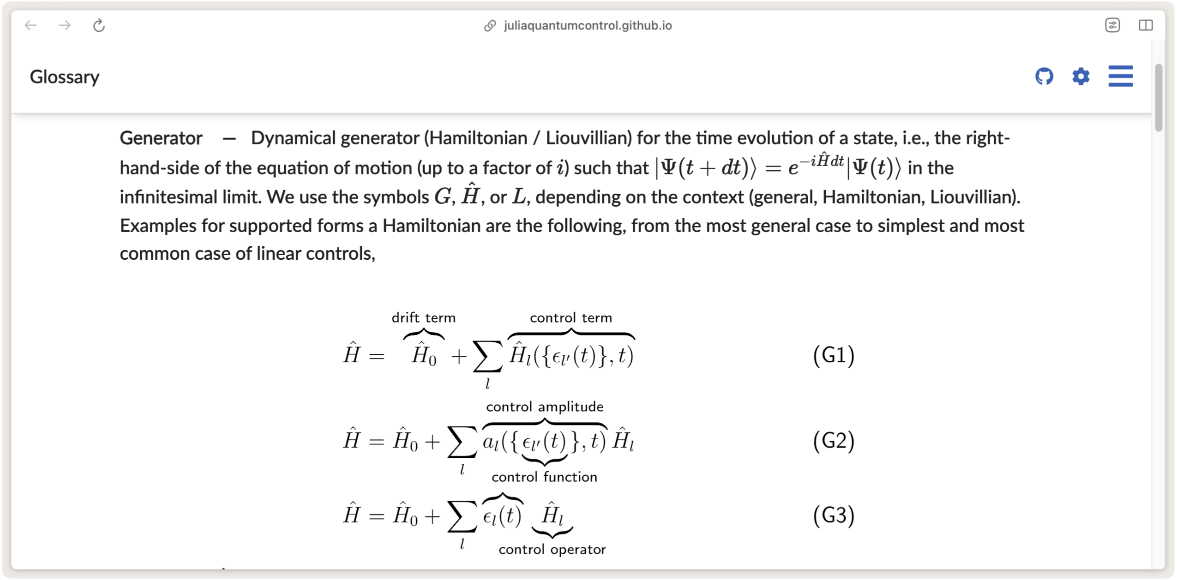
...

Initial driving field

```
[ ]: using QuantumControl.Amplitudes: ShapedAmplitude
     using QuantumControl.Shapes: flattop

     function guess_amplitudes(; T=400ns, E₀=35MHz, dt=0.1ns, t_rise=15ns)
         tlist = collect(range(0, T, step=dt))
         shape(t) = flattop(t, T=T, t_rise=t_rise)
         Ωre = ShapedAmplitude(t -> E₀, tlist; shape)
         Ωim = ShapedAmplitude(t -> 0.0, tlist; shape)
         return tlist, Ωre, Ωim
```

UNCLASSIFIED

Dynamical Generator



Glossary

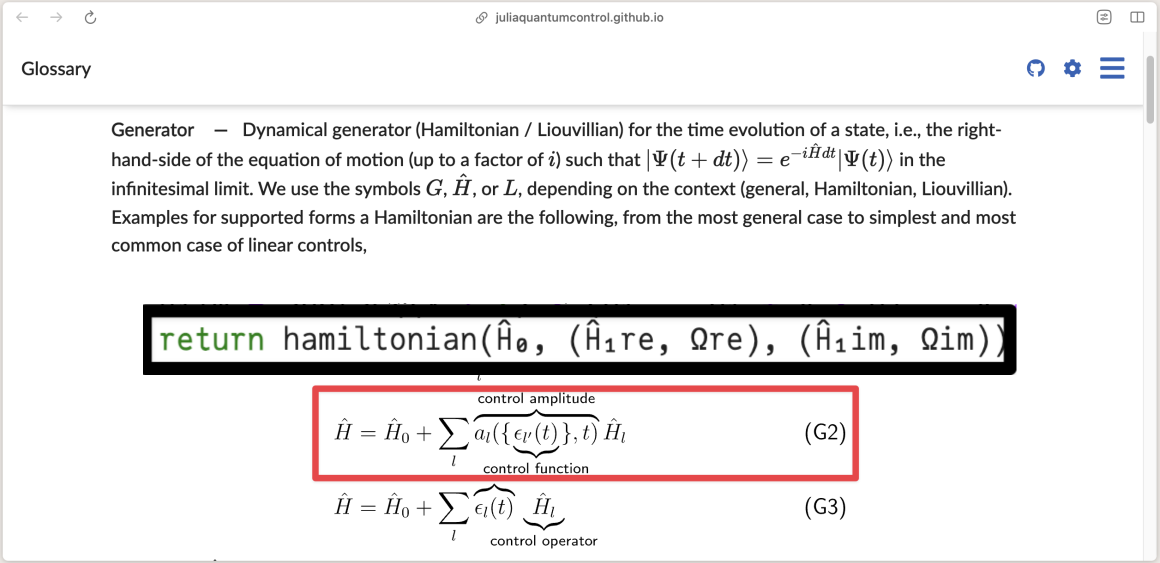
Generator — Dynamical generator (Hamiltonian / Liouvillian) for the time evolution of a state, i.e., the right-hand-side of the equation of motion (up to a factor of i) such that $|\Psi(t+dt)\rangle = e^{-i\hat{H}dt}|\Psi(t)\rangle$ in the infinitesimal limit. We use the symbols G, $\hat{H}$, or L, depending 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,

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

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

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

Dynamical Generator



juliaquantumcontrol.github.io

Glossary

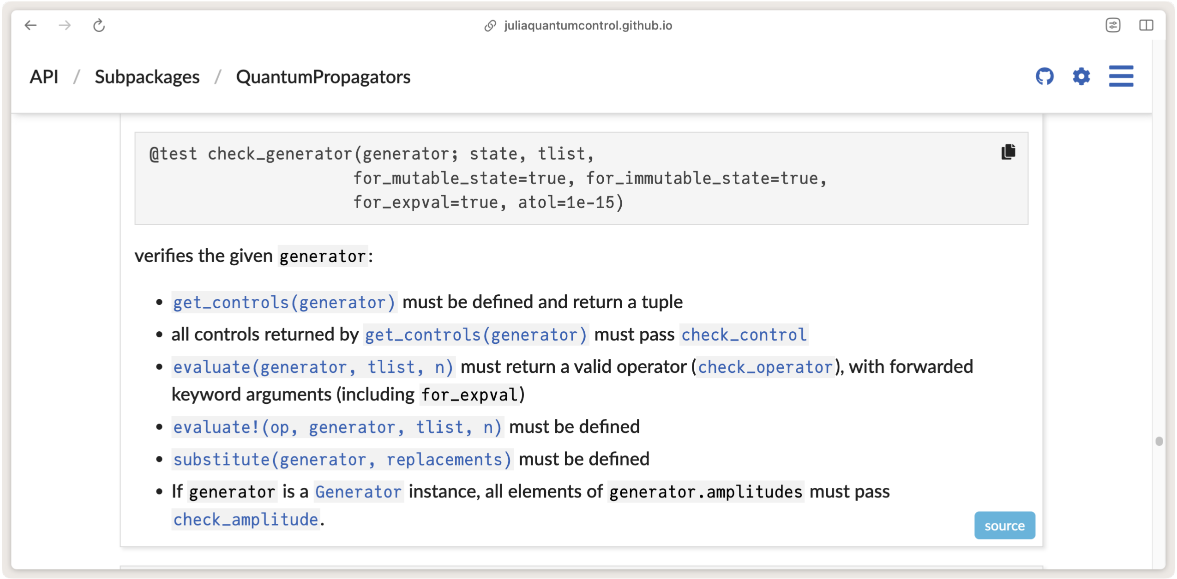
Generator — Dynamical generator (Hamiltonian / Liouvillian) for the time evolution of a state, i.e., the right-hand-side of the equation of motion (up to a factor of i) such that $|\Psi(t+dt)\rangle = e^{-i\hat{H}dt}|\Psi(t)\rangle$ in the infinitesimal limit. We use the symbols G, $\hat{H}$, or L, depending 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,

```
return hamiltonian(Ĥ₀, (Ĥ₁re, Ωre), (Ĥ₁im, Ωim))
```

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

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

Generator Interface

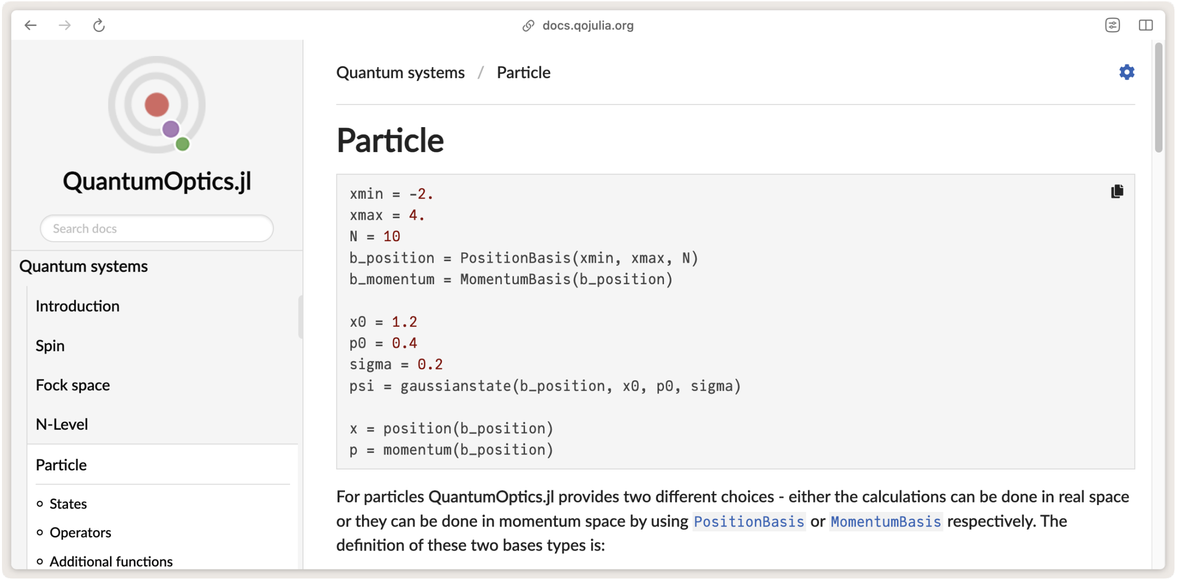

juliaquantumcontrol.github.io
API / Subpackages / QuantumPropagators
source

```
@test check_generator(generator; state, tlist,
                      for_mutable_state=true, for_immutable_state=true,
                      for_expval=true, atol=1e-15)
```

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`
- `evaluate(generator, tlist, n)` must return a valid operator (`check_operator`), with forwarded keyword arguments (including `for_expval`)
- `evaluate!(op, generator, tlist, n)` must be defined
- `substitute(generator, replacements)` must be defined
- If `generator` is a `Generator` instance, all elements of `generator.amplitudes` must pass `check_amplitude`.

QuantumControl.jl is not a modeling framework!



Quantum systems

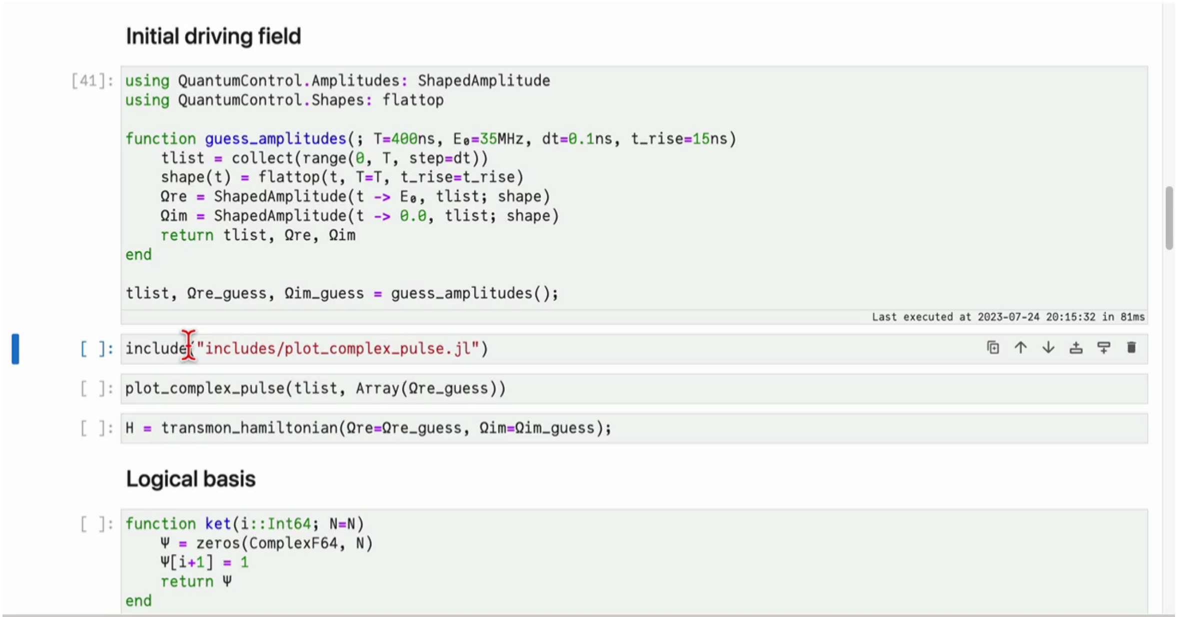
- Introduction
- Spin
- Fock space
- N-Level
- Particle
  - States
  - Operators
  - Additional functions

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:



Initial driving field

[41]:
```julia
using QuantumControl.Amplitudes: ShapedAmplitude
using QuantumControl.Shapes: flattop

function guess_amplitudes(; T=400ns, E₀=35MHz, dt=0.1ns, t_rise=15ns)
    tlist = collect(range(0, T, step=dt))
    shape(t) = flattop(t, T=T, t_rise=t_rise)
    Ωre = ShapedAmplitude(t -> E₀, tlist; shape)
    Ωim = ShapedAmplitude(t -> 0.0, tlist; shape)
    return tlist, Ωre, Ωim
end

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

Last executed at 2023-07-24 20:15:32 in 81ms

[ ]:
```julia
include("includes/plot_complex_pulse.jl")
```

[ ]:
```julia
plot_complex_pulse(tlist, Array(Ωre_guess))
```

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

Logical basis

[ ]:
```julia
function ket(i::Int64; N=N)
    Ψ = zeros(ComplexF64, N)
    Ψ[i+1] = 1
    return Ψ
end
```

```
Last executed at 2023-07-24 20:15:32 in 81ms
```

```
[42]: include("includes/plot_complex_pulse.jl")
```

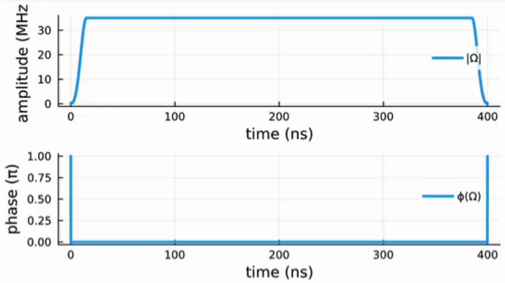
```
Last executed at 2023-07-24 20:16:27 in 11ms
```

```
[42]: plot_complex_pulse (generic function with 1 method)
```

```
[43]: plot_complex_pulse(tlist, Array(Ωre_guess))
```

```
Last executed at 2023-07-24 20:16:27 in 74ms
```

[43]:



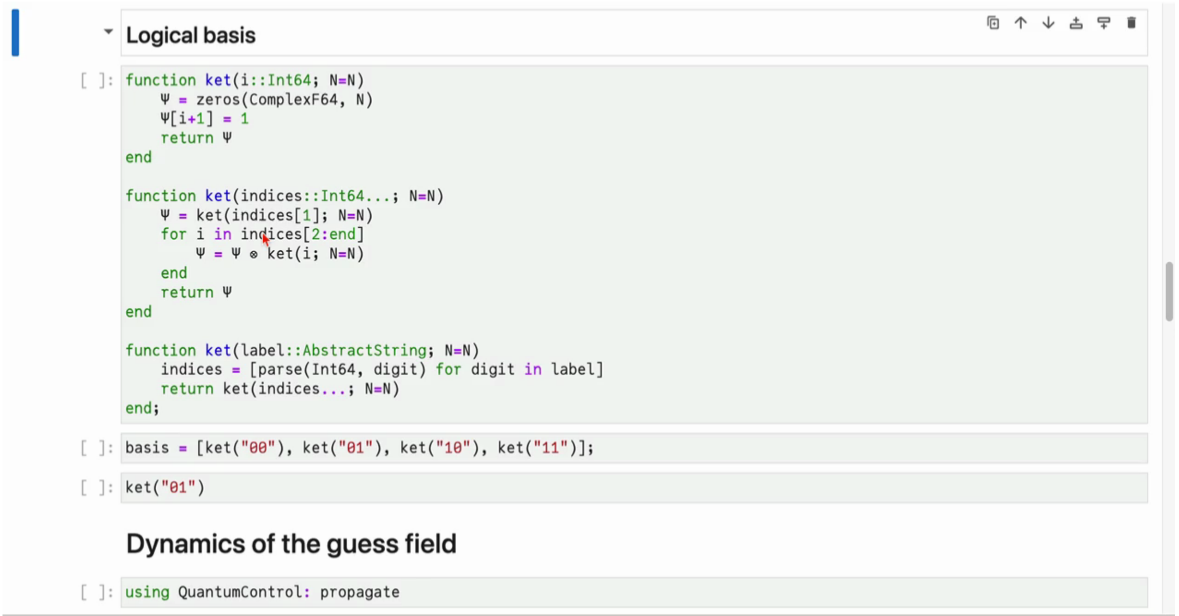
```
[44]: H = transmon_hamiltonian(Ωre=Ωre_guess, Ωim=Ωim_guess);
```

```
Last executed at 2023-07-24 20:16:31 in 65ms
```

Logical basis

```
[ ]: function ket(i::Int64; N=N)
```

UNCLASSIFIED



Logical basis

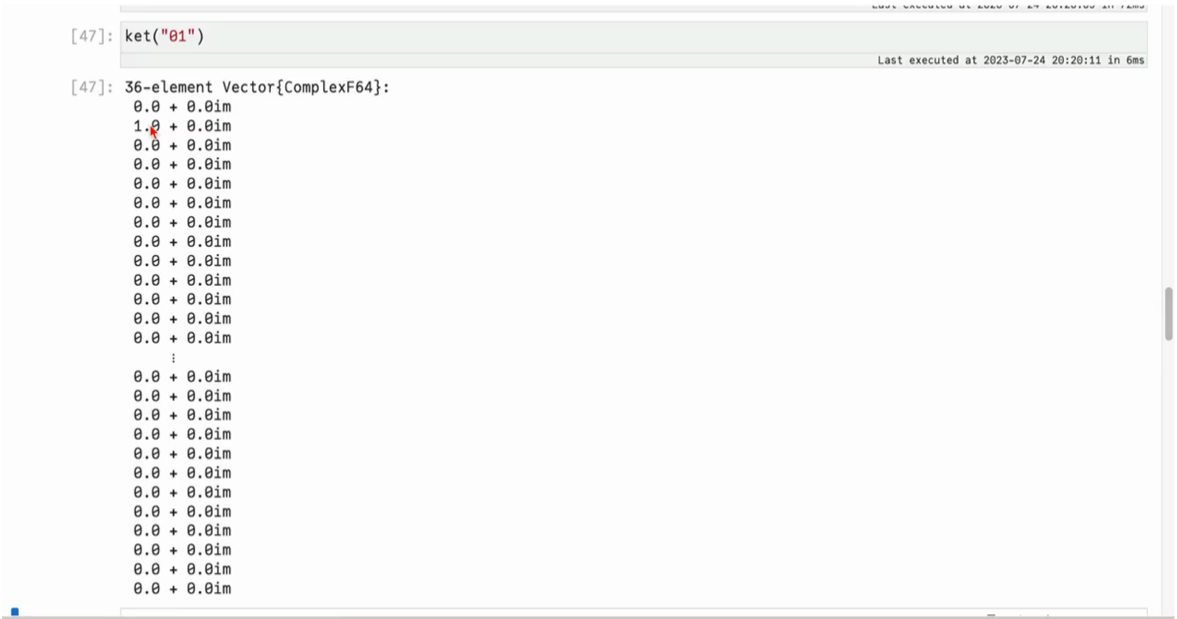
```
[ ]: function ket(i::Int64; N=N)
         Ψ = zeros(ComplexF64, N)
         Ψ[i+1] = 1
         return Ψ
     end

     function ket(indices::Int64...; N=N)
         Ψ = ket(indices[1]; N=N)
         for i in indices[2:end]
             Ψ = Ψ ⊗ ket(i; N=N)
         end
         return Ψ
     end

     function ket(label::AbstractString; N=N)
         indices = [parse(Int64, digit) for digit in label]
         return ket(indices...; N=N)
     end;
```

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

```
[ ]: ket("01")
```

Dynamics of the guess field

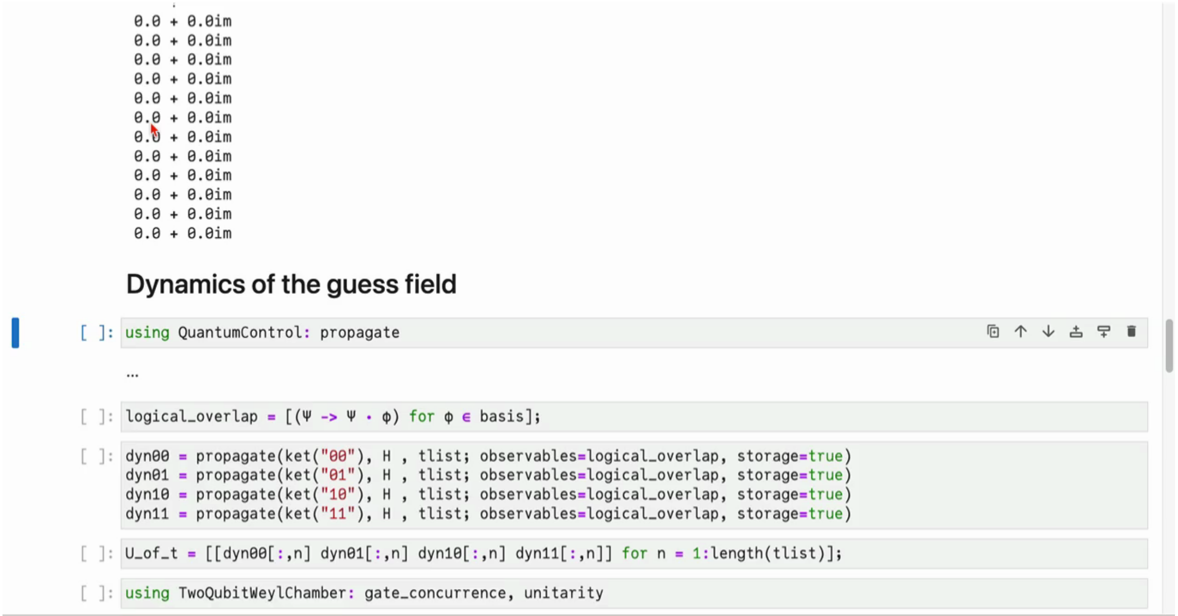
```
[ ]: using QuantumControl: propagate
```



```
[47]: ket("01")
```

Last executed at 2023-07-24 20:20:11 in 6ms

```
[47]: 36-element Vector{ComplexF64}:
       0.0 + 0.0im
       1.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
            ⋮
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
       0.0 + 0.0im
```



```
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
```

Dynamics of the guess field

```
[ ]: using QuantumControl: propagate
```

...

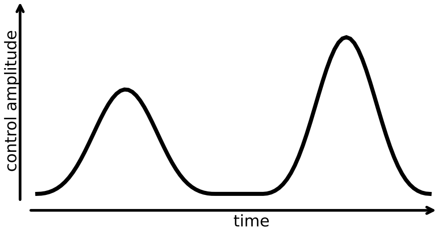
```
[ ]: logical_overlap = [(Ψ -> Ψ ⋅ ϕ) for ϕ ∈ basis];
```

```
[ ]: dyn00 = propagate(ket("00"), H , tlist; observables=logical_overlap, storage=true)
     dyn01 = propagate(ket("01"), H , tlist; observables=logical_overlap, storage=true)
     dyn10 = propagate(ket("10"), H , tlist; observables=logical_overlap, storage=true)
     dyn11 = propagate(ket("11"), H , tlist; observables=logical_overlap, storage=true)
```

```
[ ]: U_of_t = [[dyn00[:,n] dyn01[:,n] dyn10[:,n] dyn11[:,n]] for n = 1:length(tlist)];
```

```
[ ]: using TwoQubitWeylChamber: gate_concurrence, unitarity
```

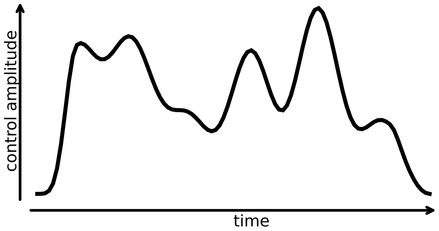
QuantumPropagators.jl



$$i\hbar \frac{\partial}{\partial t} \left|\Psi(t)\right\rangle = \hat{\mathsf{H}}(\{\epsilon_l(t)\}) \left|\Psi(t)\right\rangle$$

$$i\hbar \frac{\partial}{\partial t} \hat{\rho}(t) = \mathcal{L}(\{\epsilon_l(t)\})[\hat{\rho}(t)]$$

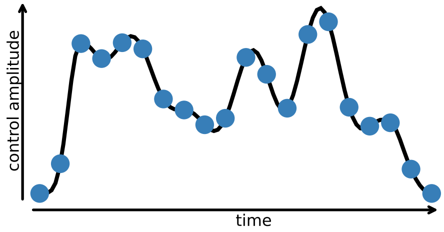
QuantumPropagators.jl



$$i\hbar\frac{\partial}{\partial t}\left|\Psi(t)\right\rangle = \hat{\mathsf{H}}(\{\epsilon_l(t)\})\left|\Psi(t)\right\rangle$$

$$i\hbar\frac{\partial}{\partial t}\hat{\rho}(t) = \mathcal{L}(\{\epsilon_l(t)\})[\hat{\rho}(t)]$$

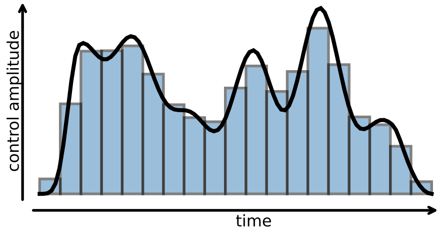
QuantumPropagators.jl



$$i\hbar\frac{\partial}{\partial t}\left|\Psi(t)\right\rangle = \hat{\mathsf{H}}(\{\epsilon_l(t)\})\left|\Psi(t)\right\rangle$$

$$i\hbar\frac{\partial}{\partial t}\hat{\rho}(t) = \mathcal{L}(\{\epsilon_l(t)\})[\hat{\rho}(t)]$$

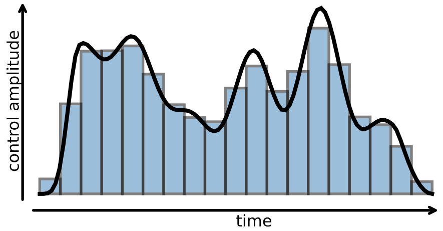
QuantumPropagators.jl



$$i\hbar \frac{\partial}{\partial t} \left|\Psi(t)\right\rangle = \hat{\mathsf{H}}(\{\epsilon_l(t)\}) \left|\Psi(t)\right\rangle$$

$$i\hbar \frac{\partial}{\partial t} \hat{\rho}(t) = \mathcal{L}(\{\epsilon_l(t)\})[\hat{\rho}(t)]$$

QuantumPropagators.jl



$$i\hbar\frac{\partial}{\partial t}\left|\Psi(t)\right\rangle = \hat{\mathrm{H}}(\{\epsilon_l(t)\})\left|\Psi(t)\right\rangle$$

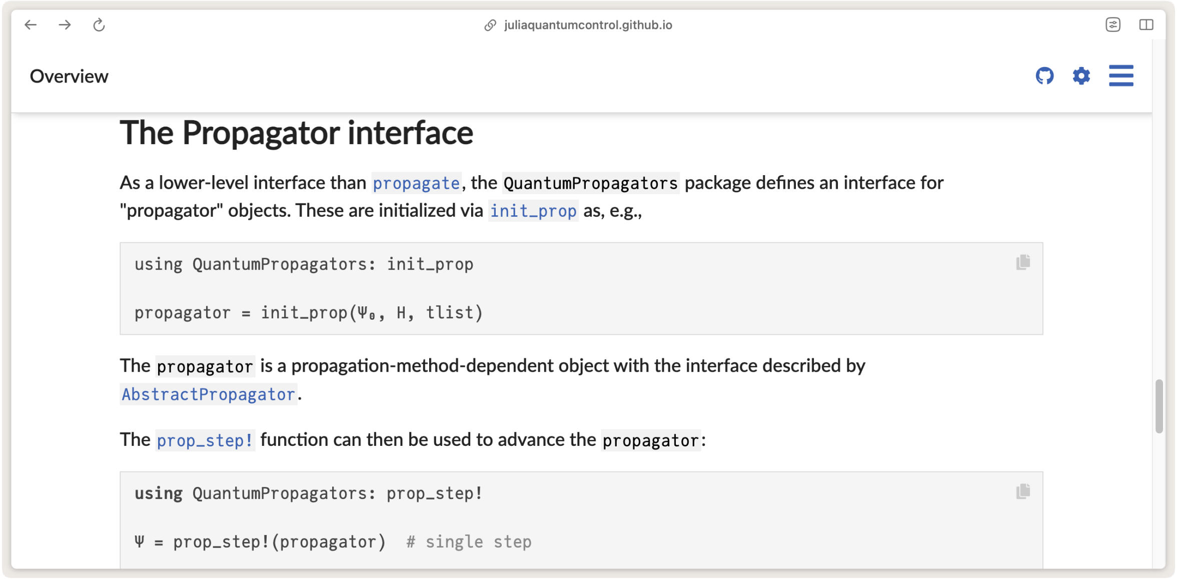
$$i\hbar\frac{\partial}{\partial t}\hat{\rho}(t) = \mathcal{L}(\{\epsilon_l(t)\})[\hat{\rho}(t)]$$

PWC propagator: $\hat{\mathrm{U}}_n = \exp[-\frac{i}{\hbar}\hat{\mathrm{H}}_n dt]$ for n'th time slice

$\Rightarrow$ evaluate $\hat{\mathrm{U}}_n\left|\Psi\right\rangle$ (or $\mathcal{U}_n[\hat{\rho}]$) as a polynomial expansion

- Hermitian Hamiltonian $\rightarrow$ Chebychev polynomials
- Non-Hermitian Hamiltonian or Liouvillian $\rightarrow$ Newton polynomials

Propagator Interface



juliaquantumcontrol.github.io

Overview

The Propagator interface

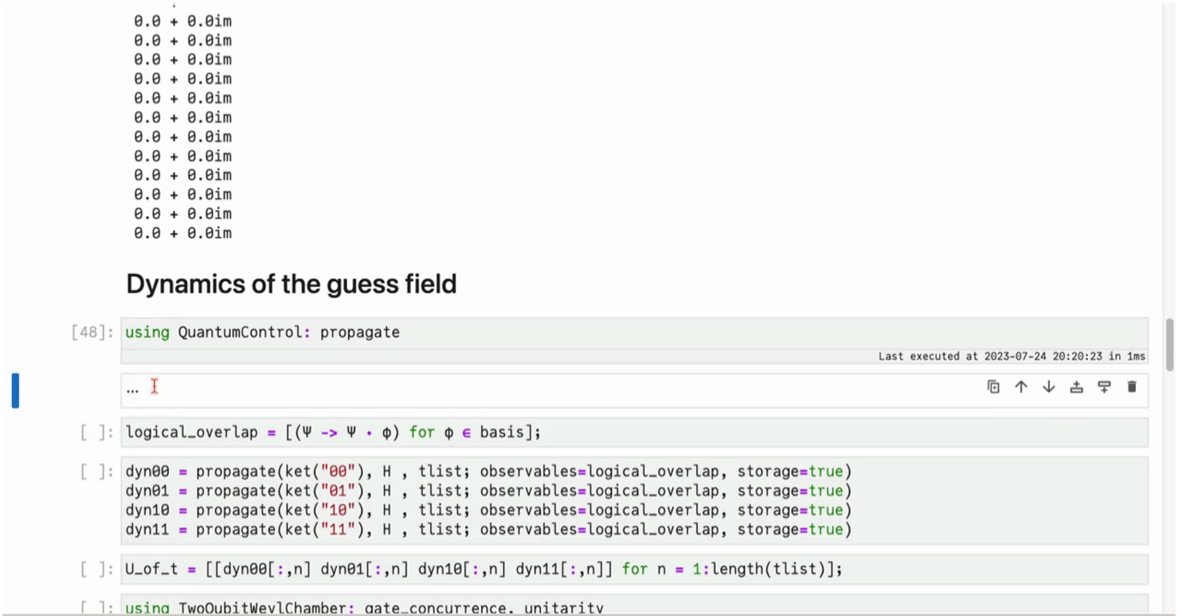
As a lower-level interface than `propagate`, the `QuantumPropagators` package defines an interface for "propagator" objects. These are initialized via `init_prop` as, e.g.,

```
using QuantumPropagators: init_prop

propagator = init_prop(Ψ₀, H, tlist)
```

The `propagator` is a propagation-method-dependent object with the interface described by `AbstractPropagator`.

The `prop_step!` function can then be used to advance the `propagator`:

```
using QuantumPropagators: prop_step!

Ψ = prop_step!(propagator)  # single step
```



```
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
 0.0 + 0.0im
```

Dynamics of the guess field

```
[48]: using QuantumControl: propagate
```

Last executed at 2023-07-24 20:20:23 in 1ms

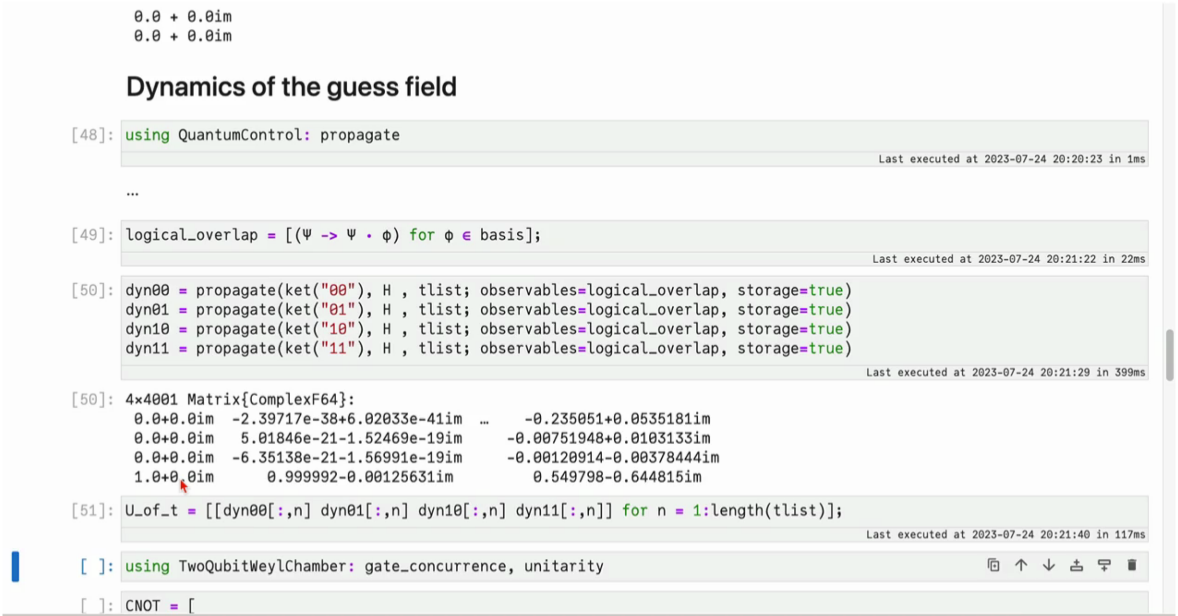
```
...
```

```
[ ]: logical_overlap = [(Ψ -> Ψ ⋅ ϕ) for ϕ ∈ basis];
```

```
[ ]: dyn00 = propagate(ket("00"), H , tlist; observables=logical_overlap, storage=true)
     dyn01 = propagate(ket("01"), H , tlist; observables=logical_overlap, storage=true)
     dyn10 = propagate(ket("10"), H , tlist; observables=logical_overlap, storage=true)
     dyn11 = propagate(ket("11"), H , tlist; observables=logical_overlap, storage=true)
```

```
[ ]: U_of_t = [[dyn00[:,n] dyn01[:,n] dyn10[:,n] dyn11[:,n]] for n = 1:length(tlist)];
```

```
[ ]: using TwoQubitWeylChamber: gate_concurrence, unitarity
```



```
 0.0 + 0.0im
 0.0 + 0.0im
```

Dynamics of the guess field

```
[48]: using QuantumControl: propagate
```

`Last executed at 2023-07-24 20:20:23 in 1ms`

…

```
[49]: logical_overlap = [(Ψ -> Ψ ⋅ ϕ) for ϕ ∈ basis];
```

`Last executed at 2023-07-24 20:21:22 in 22ms`

```
[50]: dyn00 = propagate(ket("00"), H , tlist; observables=logical_overlap, storage=true)
      dyn01 = propagate(ket("01"), H , tlist; observables=logical_overlap, storage=true)
      dyn10 = propagate(ket("10"), H , tlist; observables=logical_overlap, storage=true)
      dyn11 = propagate(ket("11"), H , tlist; observables=logical_overlap, storage=true)
```

`Last executed at 2023-07-24 20:21:29 in 399ms`

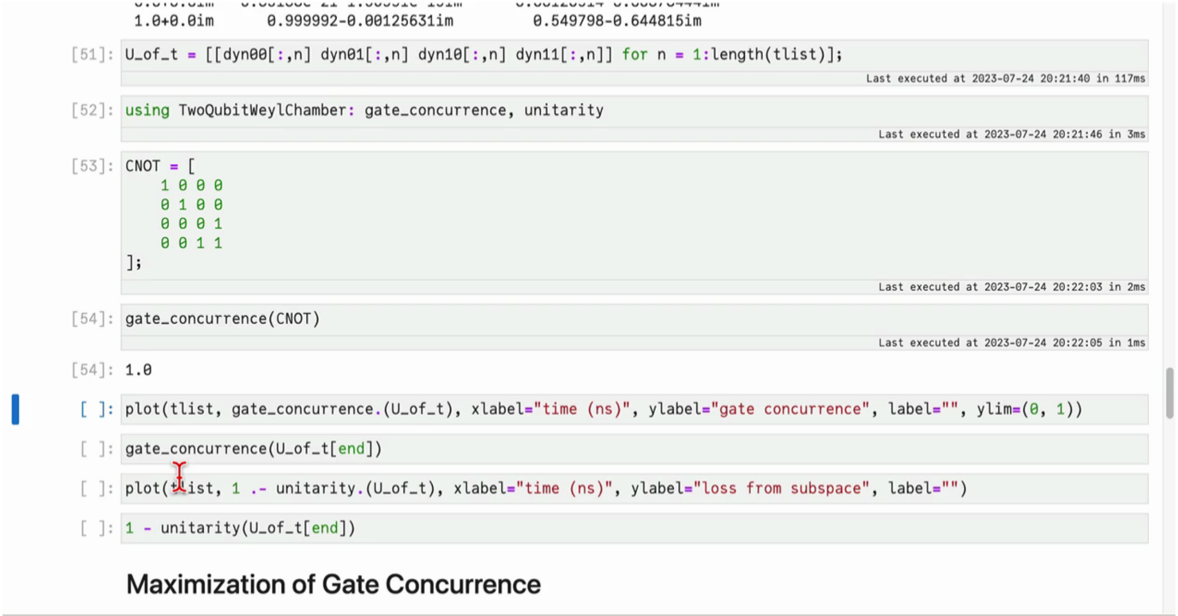
```
[50]: 4×4001 Matrix{ComplexF64}:
       0.0+0.0im  -2.39717e-38+6.02033e-41im  …    -0.235051+0.0535181im
       0.0+0.0im   5.01846e-21-1.52469e-19im     -0.00751948+0.0103133im
       0.0+0.0im  -6.35138e-21-1.56991e-19im     -0.00120914-0.00378444im
       1.0+0.0im      0.999992-0.00125631im         0.549798-0.644815im
```

```
[51]: U_of_t = [[dyn00[:,n] dyn01[:,n] dyn10[:,n] dyn11[:,n]] for n = 1:length(tlist)];
```

`Last executed at 2023-07-24 20:21:40 in 117ms`

```
[ ]: using TwoQubitWeylChamber: gate_concurrence, unitarity
```

```
[ ]: CNOT = [
```

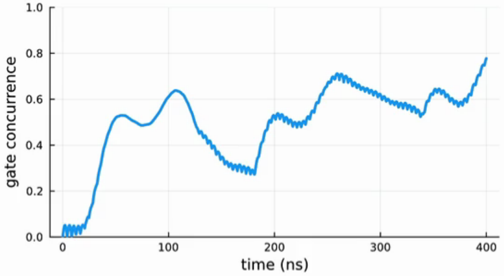


```
 1.0+0.0im      0.999992-0.00125631im        0.549798-0.644815im
```

```
[51]: U_of_t = [[dyn00[:,n] dyn01[:,n] dyn10[:,n] dyn11[:,n]] for n = 1:length(tlist)];
                                                  Last executed at 2023-07-24 20:21:40 in 117ms

[52]: using TwoQubitWeylChamber: gate_concurrence, unitarity
                                                  Last executed at 2023-07-24 20:21:46 in 3ms

[53]: CNOT = [
          1 0 0 0
          0 1 0 0
          0 0 0 1
          0 0 1 1
      ];
                                                  Last executed at 2023-07-24 20:22:03 in 2ms

[54]: gate_concurrence(CNOT)
                                                  Last executed at 2023-07-24 20:22:05 in 1ms

[54]: 1.0

[ ]: plot(tlist, gate_concurrence.(U_of_t), xlabel="time (ns)", ylabel="gate concurrence", label="", ylim=(0, 1))

[ ]: gate_concurrence(U_of_t[end])

[ ]: plot(tlist, 1 .- unitarity.(U_of_t), xlabel="time (ns)", ylabel="loss from subspace", label="")

[ ]: 1 - unitarity(U_of_t[end])
```

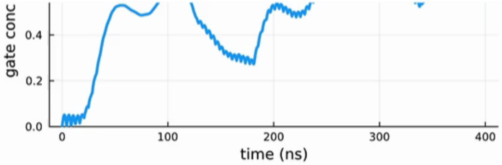
Maximization of Gate Concurrence

```
                                                    Last executed at 2023-07-24 20:22:03 in 2ms
[54]: gate_concurrence(CNOT)
                                                    Last executed at 2023-07-24 20:22:05 in 1ms
[54]: 1.0

[55]: plot(tlist, gate_concurrence.(U_of_t), xlabel="time (ns)", ylabel="gate concurrence", label="", ylim=(0, 1))
                                                    Last executed at 2023-07-24 20:22:10 in 80ms
[55]:
```

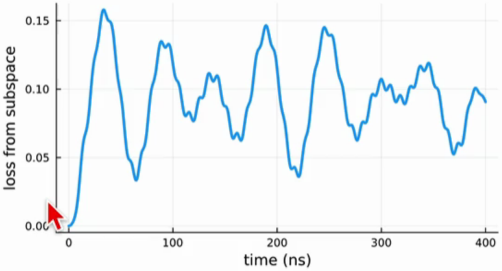


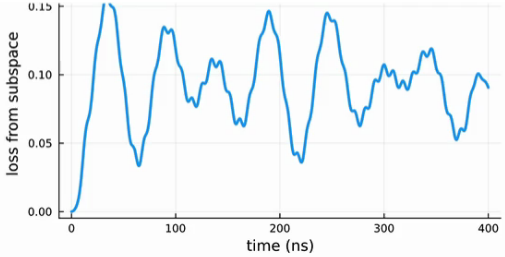
```
[56]: gate_concurrence(U_of_t[end])
                                                    Last executed at 2023-07-24 20:22:20 in 2ms
[56]: 0.7773116198529164

[ ]: plot(tlist, 1 .- unitarity.(U_of_t), xlabel="time (ns)", ylabel="loss from subspace", label="")

[ ]: 1 - unitarity(U_of_t[end])
```

UNCLASSIFIED



```
[56]: gate_concurrence(U_of_t[end])
                                        Last executed at 2023-07-24 20:22:20 in 2ms

[56]: 0.7773116198529164

[57]: plot(tlist, 1 .- unitarity.(U_of_t), xlabel="time (ns)", ylabel="loss from subspace", label="")
                                        Last executed at 2023-07-24 20:22:35 in 37ms

[57]:
```

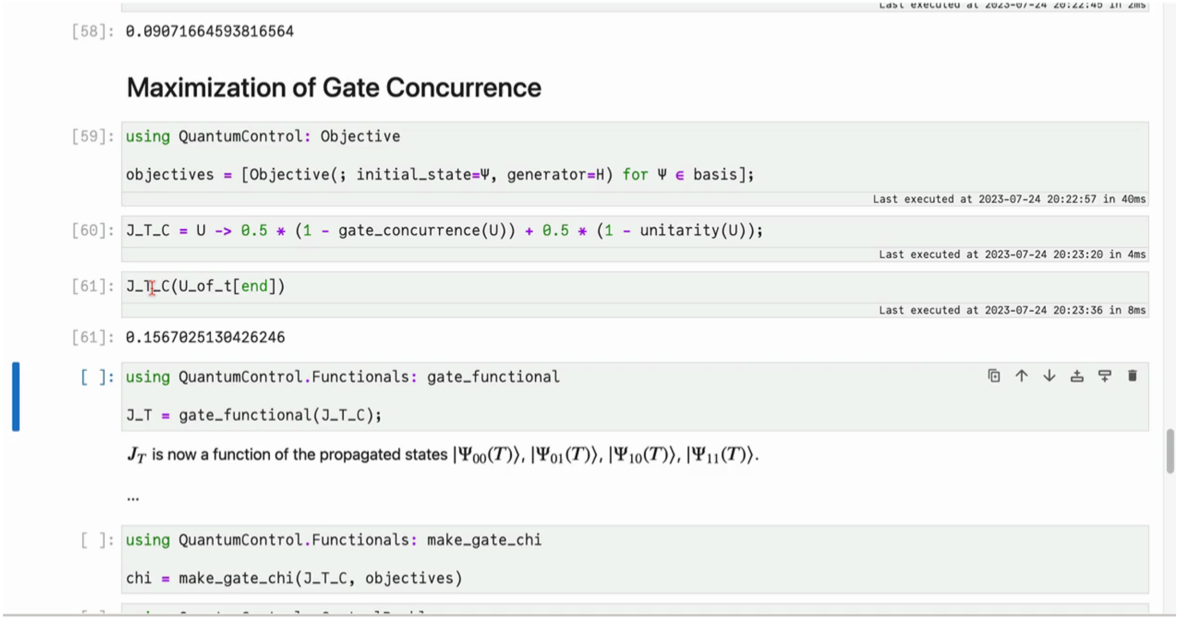




```
[58]: 1 - unitarity(U_of_t[end])
                                        Last executed at 2023-07-24 20:22:45 in 2ms
[58]: 0.09071664593816564
```

Maximization of Gate Concurrence

```
[59]: using QuantumControl: Objective

      objectives = [Objective(; initial_state=Ψ, generator=H) for Ψ ∈ basis];
                                        Last executed at 2023-07-24 20:22:57 in 40ms

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

[ ]: J_T_C(U_of_t[end])
```



```
Last executed at 2023-07-24 20:22:45 in 2ms
[58]: 0.09071664593816564
```

Maximization of Gate Concurrence

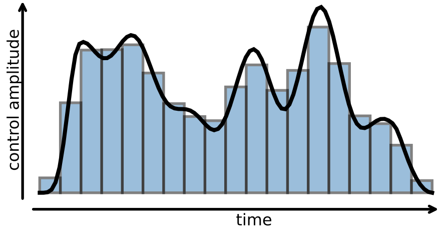
```
[59]: using QuantumControl: Objective

      objectives = [Objective(; initial_state=Ψ, generator=H) for Ψ ∈ basis];
                                              Last executed at 2023-07-24 20:22:57 in 40ms

[60]: J_T_C = U -> 0.5 * (1 - gate_concurrence(U)) + 0.5 * (1 - unitarity(U));
                                              Last executed at 2023-07-24 20:23:20 in 4ms

[61]: J_T_C(U_of_t[end])
                                              Last executed at 2023-07-24 20:23:36 in 8ms

[61]: 0.1567025130426246

[ ]: using QuantumControl.Functionals: gate_functional

     J_T = gate_functional(J_T_C);
```

J_T is now a function of the propagated states $|\Psi_{00}(T)\rangle$, $|\Psi_{01}(T)\rangle$, $|\Psi_{10}(T)\rangle$, $|\Psi_{11}(T)\rangle$.

...

```
[ ]: using QuantumControl.Functionals: make_gate_chi

     chi = make_gate_chi(J_T_C, objectives)
```

Gradient-based optimal control

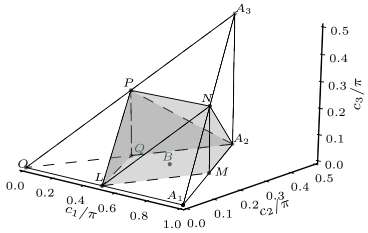


- Control parameters: discretized pulse values ϵ_{nl}
- Gradient $\nabla J_T = \frac{\partial J_T}{\partial \epsilon_{nl}}$
- Tune controls in the direction of the gradient

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

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

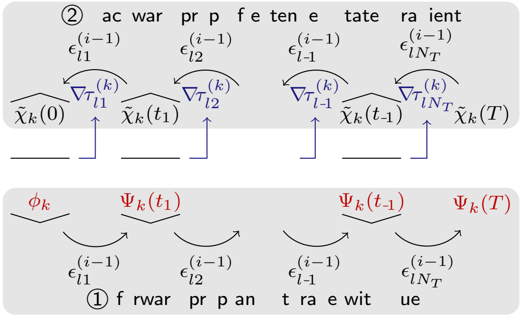
Childs *et al.* Phys. Rev. A 68, 052311 (2003)



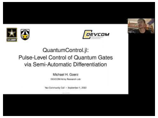
Not analytic!

Semi-automatic differentiation

$$
\begin{aligned}
\nabla J_T &= \frac{\partial J_T(\{\Psi_k(T)\})}{\partial \epsilon_{nl}} \\
&= 2\mathrm{Re}\left[\sum_k \underbrace{\frac{\partial J_T}{\partial \left|\Psi_k(T)\right\rangle}}_{\equiv \langle \chi_k |} \frac{\partial \left|\Psi_k(T)\right\rangle}{\partial \epsilon_{nl}}\right] \\
&= 2\mathrm{Re}\left[\sum_k \frac{\partial}{\partial \epsilon_{nl}} \langle \chi_k(T) | \Psi_k(T) \rangle\right]
\end{aligned}
$$

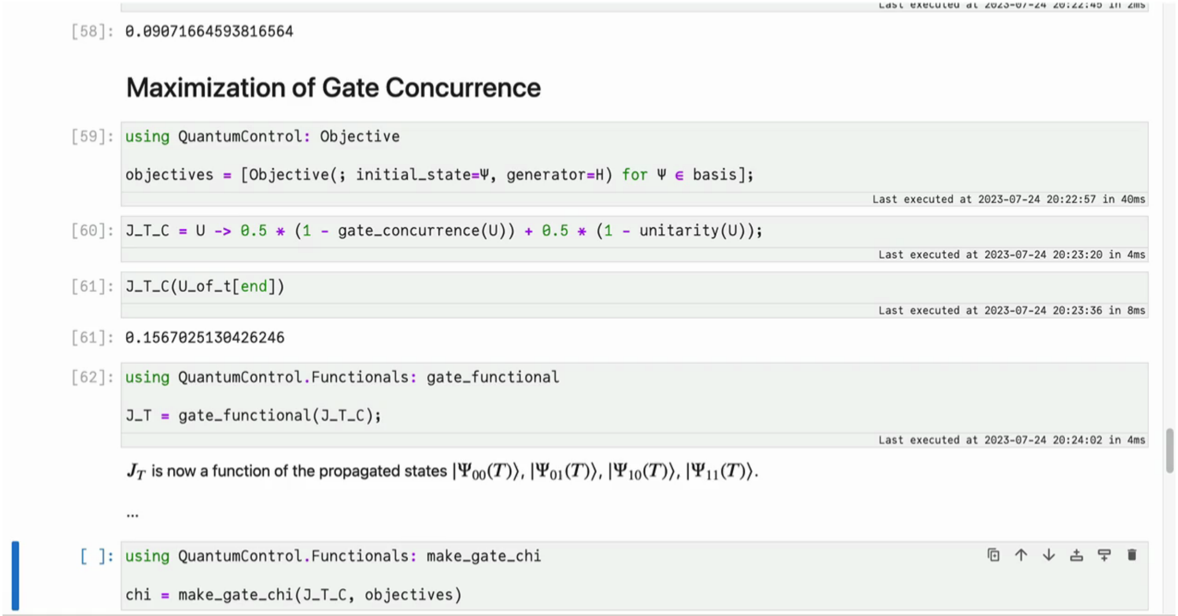

② ac war pr p f e ten e tate ra ient
$\epsilon_{l1}^{(i-1)}$
$\epsilon_{l2}^{(i-1)}$
$\epsilon_{l\text{-}1}^{(i-1)}$
$\epsilon_{lN_T}^{(i-1)}$
$\nabla\tau_{l1}^{(k)}$
$\nabla\tau_{l2}^{(k)}$
$\nabla\tau_{l\text{-}1}^{(k)}$
$\nabla\tau_{lN_T}^{(k)}$
$\tilde{\chi}_k(0)$
$\tilde{\chi}_k(t_1)$
$\tilde{\chi}_k(t_{\text{-}1})$
$\tilde{\chi}_k(T)$
ϕ_k
$\Psi_k(t_1)$
$\Psi_k(t_{\text{-}1})$
$\Psi_k(T)$
$\epsilon_{l1}^{(i-1)}$
$\epsilon_{l2}^{(i-1)}$
$\epsilon_{l\text{-}1}^{(i-1)}$
$\epsilon_{lN_T}^{(i-1)}$
① f rwar pr p an t ra e wit ue

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


QuantumControl.jl:
Pulse-Level Control of Quantum Gates
via Semi-Automatic Differentiation

Michael H. Goerz

Yao Community Seminar:
`https://youtu.be/MQCILD2P89c`



```
Last executed at 2023-07-24 20:22:45 in 2ms
[58]: 0.09071664593816564
```

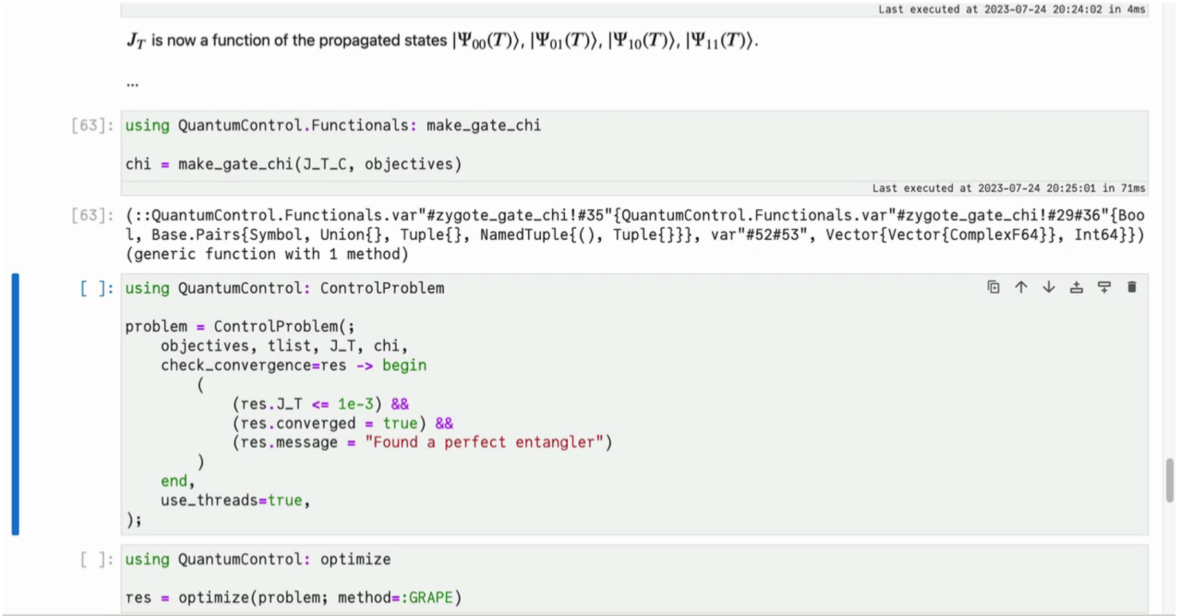
Maximization of Gate Concurrence

```
[59]: using QuantumControl: Objective

      objectives = [Objective(; initial_state=Ψ, generator=H) for Ψ ∈ basis];
                                                Last executed at 2023-07-24 20:22:57 in 40ms
[60]: J_T_C = U -> 0.5 * (1 - gate_concurrence(U)) + 0.5 * (1 - unitarity(U));
                                                Last executed at 2023-07-24 20:23:20 in 4ms
[61]: J_T_C(U_of_t[end])
                                                Last executed at 2023-07-24 20:23:36 in 8ms
[61]: 0.1567025130426246
[62]: using QuantumControl.Functionals: gate_functional

      J_T = gate_functional(J_T_C);
                                                Last executed at 2023-07-24 20:24:02 in 4ms
```

J_T is now a function of the propagated states $|\Psi_{00}(T)\rangle$, $|\Psi_{01}(T)\rangle$, $|\Psi_{10}(T)\rangle$, $|\Psi_{11}(T)\rangle$.

...

```
[ ]: using QuantumControl.Functionals: make_gate_chi

     chi = make_gate_chi(J_T_C, objectives)
```

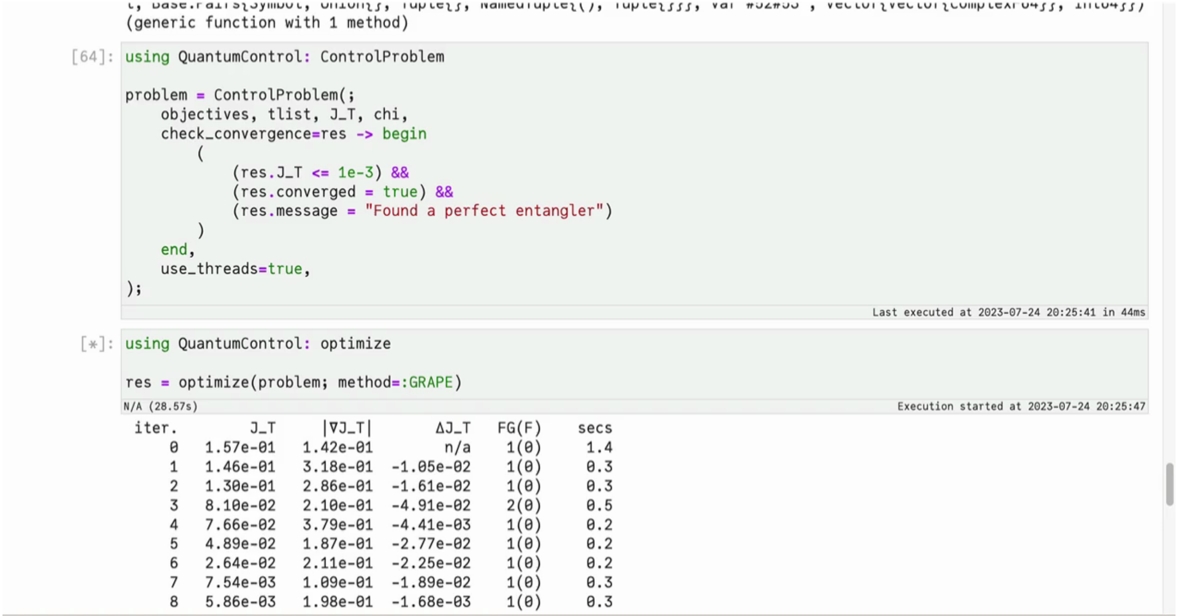


```
Last executed at 2023-07-24 20:24:02 in 4ms
```

J_T is now a function of the propagated states $|\Psi_{00}(T)\rangle$, $|\Psi_{01}(T)\rangle$, $|\Psi_{10}(T)\rangle$, $|\Psi_{11}(T)\rangle$.

...

```
[63]: using QuantumControl.Functionals: make_gate_chi

      chi = make_gate_chi(J_T_C, objectives)
                                   Last executed at 2023-07-24 20:25:01 in 71ms

[63]: (::QuantumControl.Functionals.var"#zygote_gate_chi!#35"{QuantumControl.Functionals.var"#zygote_gate_chi!#29#36"{Boo
      l, Base.Pairs{Symbol, Union{}, Tuple{}, NamedTuple{(), Tuple{}}}, var"#52#53", Vector{Vector{ComplexF64}}, Int64}})
      (generic function with 1 method)

[ ]:  using QuantumControl: ControlProblem

      problem = ControlProblem(;
          objectives, tlist, J_T, chi,
          check_convergence=res -> begin
              (
                  (res.J_T <= 1e-3) &&
                  (res.converged = true) &&
                  (res.message = "Found a perfect entangler")
              )
          end,
          use_threads=true,
      );

[ ]:  using QuantumControl: optimize

      res = optimize(problem; method=:GRAPE)
```



```
t, Base.Pairs{Symbol, Union{}, Tuple{}, NamedTuple{(), Tuple{}}}, var"#52#53", Vector{Vector{ComplexF64}}, Int64}}
(generic function with 1 method)
```

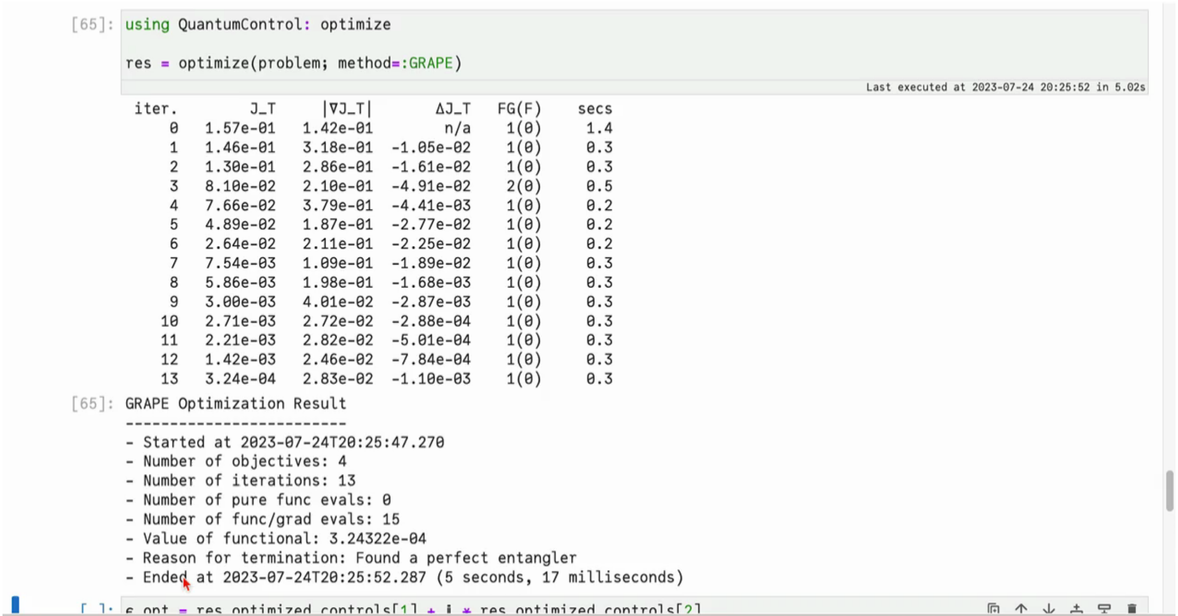
[64]:
```
using QuantumControl: ControlProblem

problem = ControlProblem(;
    objectives, tlist, J_T, chi,
    check_convergence=res -> begin
        (
            (res.J_T <= 1e-3) &&
            (res.converged = true) &&
            (res.message = "Found a perfect entangler")
        )
    end,
    use_threads=true,
);
```

Last executed at 2023-07-24 20:25:41 in 44ms

[*]:
```
using QuantumControl: optimize

res = optimize(problem; method=:GRAPE)
```

N/A (28.57s) Execution started at 2023-07-24 20:25:47

```
 iter.        J_T      |∇J_T|       ΔJ_T    FG(F)     secs
     0   1.57e-01   1.42e-01        n/a     1(0)      1.4
     1   1.46e-01   3.18e-01  -1.05e-02     1(0)      0.3
     2   1.30e-01   2.86e-01  -1.61e-02     1(0)      0.3
     3   8.10e-02   2.10e-01  -4.91e-02     2(0)      0.5
     4   7.66e-02   3.79e-01  -4.41e-03     1(0)      0.2
     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)      0.3
     8   5.86e-03   1.98e-01  -1.68e-03     1(0)      0.3
```



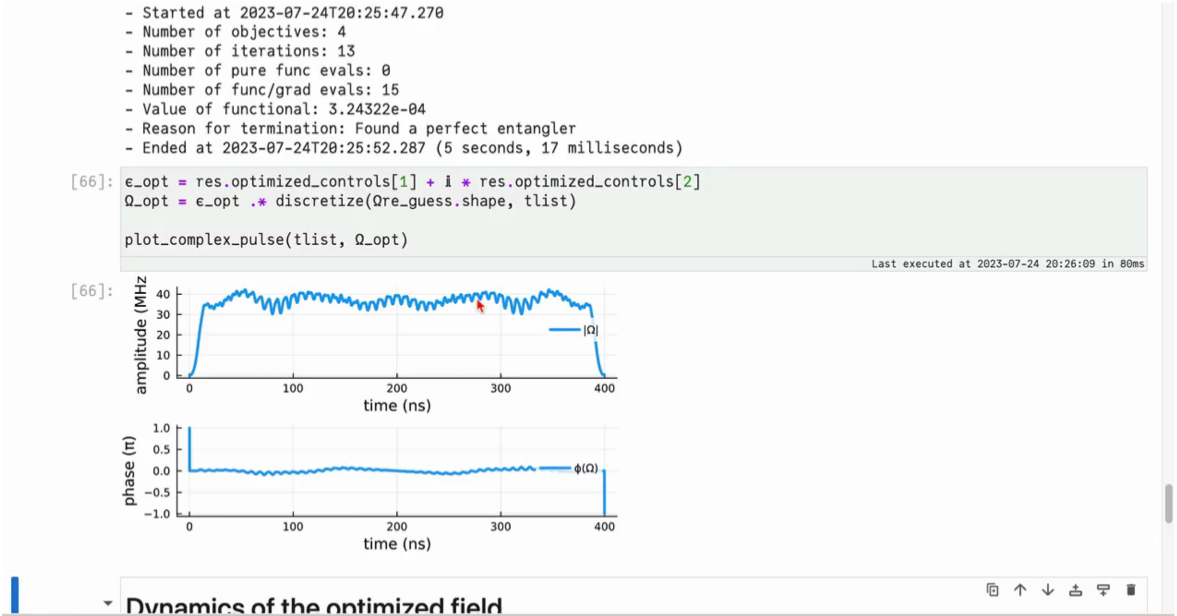
```
[65]: using QuantumControl: optimize

      res = optimize(problem; method=:GRAPE)
```

Last executed at 2023-07-24 20:25:52 in 5.02s

```
 iter.        J_T      |∇J_T|       ΔJ_T   FG(F)    secs
     0   1.57e-01   1.42e-01        n/a    1(0)     1.4
     1   1.46e-01   3.18e-01  -1.05e-02    1(0)     0.3
     2   1.30e-01   2.86e-01  -1.61e-02    1(0)     0.3
     3   8.10e-02   2.10e-01  -4.91e-02    2(0)     0.5
     4   7.66e-02   3.79e-01  -4.41e-03    1(0)     0.2
     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)     0.3
     8   5.86e-03   1.98e-01  -1.68e-03    1(0)     0.3
     9   3.00e-03   4.01e-02  -2.87e-03    1(0)     0.3
    10   2.71e-03   2.72e-02  -2.88e-04    1(0)     0.3
    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)     0.3
    13   3.24e-04   2.83e-02  -1.10e-03    1(0)     0.3
```

```
[65]: GRAPE Optimization Result
      -------------------------
      - Started at 2023-07-24T20:25:47.270
      - 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 2023-07-24T20:25:52.287 (5 seconds, 17 milliseconds)
```

```
[ ]: ϵ_opt = res.optimized_controls[1] + 𝕚 * res.optimized_controls[2]
```



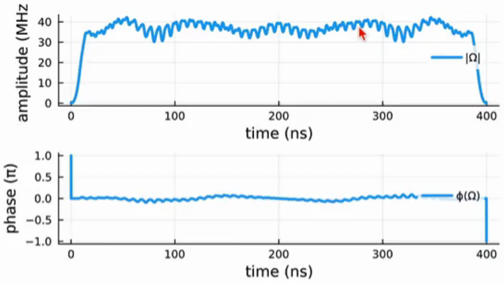
```
- Started at 2023-07-24T20:25:47.270
- Number of objectives: 4
- Number of iterations: 13
- Number of pure func evals: 0
- Number of func/grad evals: 15
- Value of functional: 3.24322e-04
- Reason for termination: Found a perfect entangler
- Ended at 2023-07-24T20:25:52.287 (5 seconds, 17 milliseconds)
```

[66]:

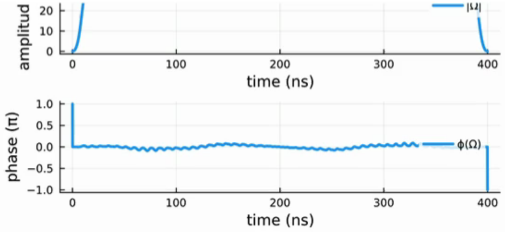
```
ϵ_opt = res.optimized_controls[1] + 𝕚 * res.optimized_controls[2]
Ω_opt = ϵ_opt .* discretize(Ωre_guess.shape, tlist)

plot_complex_pulse(tlist, Ω_opt)
```

Last executed at 2023-07-24 20:26:09 in 80ms

[66]:



Dynamics of the optimized field

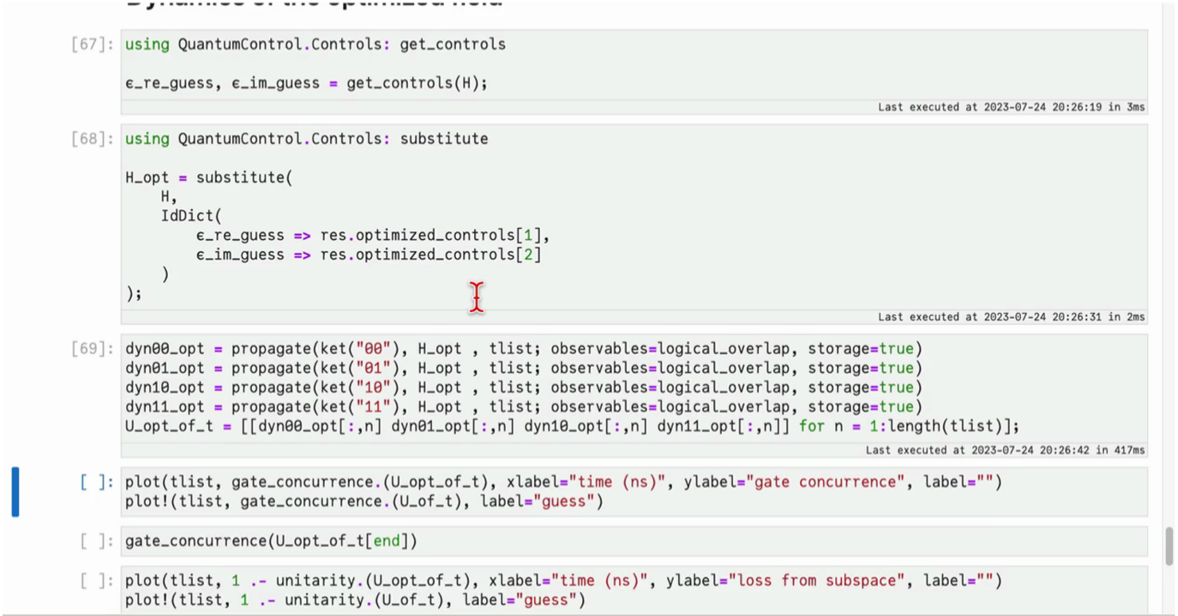


Dynamics of the optimized field ¶

[67]:
```julia
using QuantumControl.Controls: get_controls

ϵ_re_guess, ϵ_im_guess = get_controls(H);
```

```
Last executed at 2023-07-24 20:26:19 in 3ms
```

[ ]:
```julia
using QuantumControl.Controls: substitute

H_opt = substitute(
    H,
    IdDict(
        ϵ_re_guess => res.optimized_controls[1],
        ϵ_im_guess => res.optimized_controls[2]
    )
);
```



[67]:
```julia
using QuantumControl.Controls: get_controls

ϵ_re_guess, ϵ_im_guess = get_controls(H);
```
Last executed at 2023-07-24 20:26:19 in 3ms

[68]:
```julia
using QuantumControl.Controls: substitute

H_opt = substitute(
    H,
    IdDict(
        ϵ_re_guess => res.optimized_controls[1],
        ϵ_im_guess => res.optimized_controls[2]
    )
);
```
Last executed at 2023-07-24 20:26:31 in 2ms

[69]:
```julia
dyn00_opt = propagate(ket("00"), H_opt , tlist; observables=logical_overlap, storage=true)
dyn01_opt = propagate(ket("01"), H_opt , tlist; observables=logical_overlap, storage=true)
dyn10_opt = propagate(ket("10"), H_opt , tlist; observables=logical_overlap, storage=true)
dyn11_opt = propagate(ket("11"), H_opt , tlist; observables=logical_overlap, storage=true)
U_opt_of_t = [[dyn00_opt[:,n] dyn01_opt[:,n] dyn10_opt[:,n] dyn11_opt[:,n]] for n = 1:length(tlist)];
```
Last executed at 2023-07-24 20:26:42 in 417ms

[ ]:
```julia
plot(tlist, gate_concurrence.(U_opt_of_t), xlabel="time (ns)", ylabel="gate concurrence", label="")
plot!(tlist, gate_concurrence.(U_of_t), label="guess")
```

[ ]:
```julia
gate_concurrence(U_opt_of_t[end])
```

[ ]:
```julia
plot(tlist, 1 .- unitarity.(U_opt_of_t), xlabel="time (ns)", ylabel="loss from subspace", label="")
plot!(tlist, 1 .- unitarity.(U_of_t), label="guess")
```

```
    )
);
```

```
Last executed at 2023-07-24 20:26:31 in 2ms
```

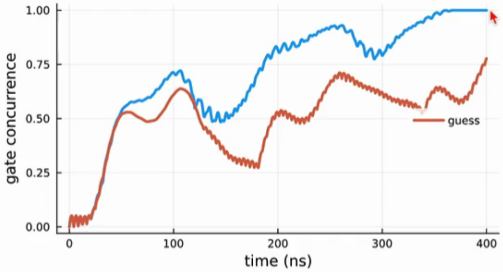
```
[69]: dyn00_opt = propagate(ket("00"), H_opt , tlist; observables=logical_overlap, storage=true)
      dyn01_opt = propagate(ket("01"), H_opt , tlist; observables=logical_overlap, storage=true)
      dyn10_opt = propagate(ket("10"), H_opt , tlist; observables=logical_overlap, storage=true)
      dyn11_opt = propagate(ket("11"), H_opt , tlist; observables=logical_overlap, storage=true)
      U_opt_of_t = [[dyn00_opt[:,n] dyn01_opt[:,n] dyn10_opt[:,n] dyn11_opt[:,n]] for n = 1:length(tlist)];
```

```
Last executed at 2023-07-24 20:26:42 in 417ms
```

```
[70]: plot(tlist, gate_concurrence.(U_opt_of_t), xlabel="time (ns)", ylabel="gate concurrence", label="")
      plot!(tlist, gate_concurrence.(U_of_t), label="guess")
```

```
Last executed at 2023-07-24 20:26:47 in 123ms
```

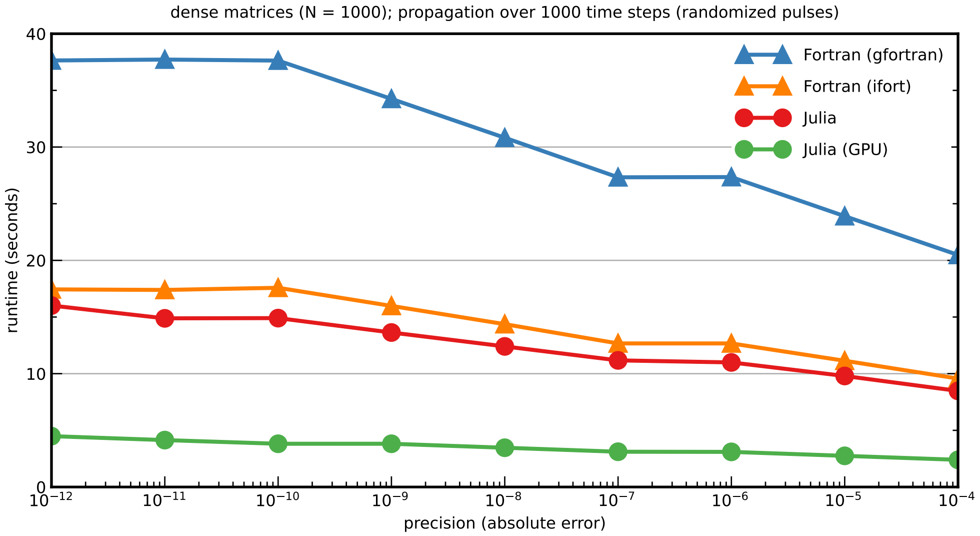
[70]:



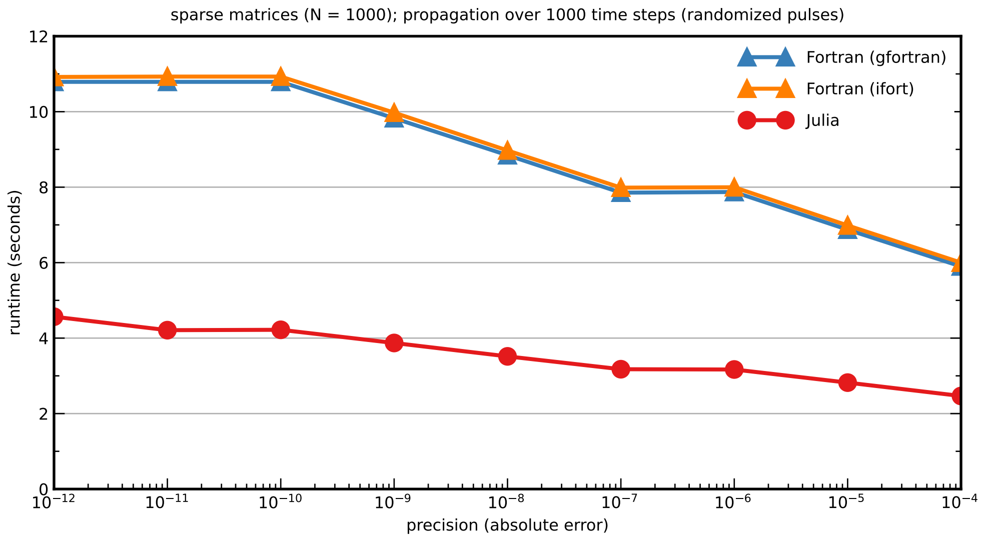
```
[ ]: gate_concurrence(U_opt_of_t[end])
```

Performance

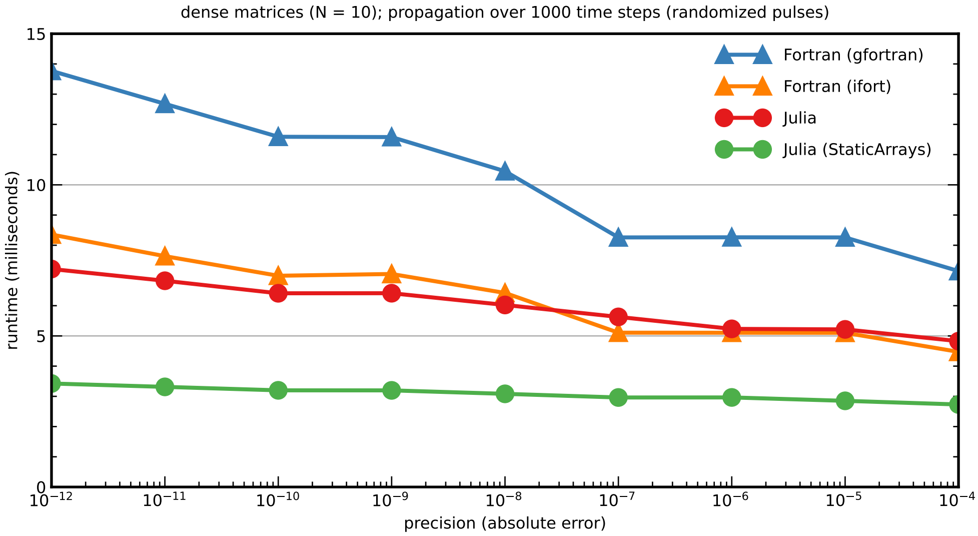
Benchmark for Chebychev Propagator – Large Hilbert Space



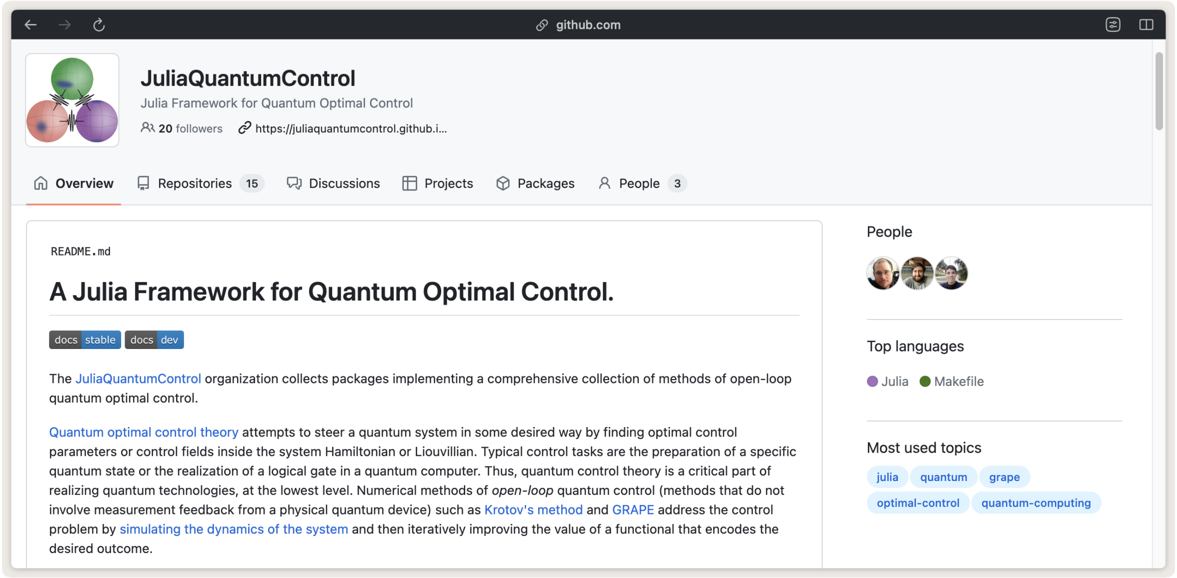
Benchmark for Chebychev Propagator – Large Hilbert Space (sparse)



Benchmark for Chebychev Propagator – Small Hilbert Space



Conclusions



JuliaQuantumControl

Julia Framework for Quantum Optimal Control

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

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.

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

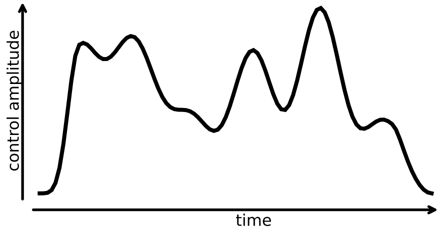
Top languages

● Julia ● Makefile

Most used topics

julia quantum grape optimal-control quantum-computing

Outlook



piecewise-constant pulses
$\Rightarrow$ parametrized continuous controls

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

- Adapt to experimental constraints on controls
- No PWC error: use DifferentialEquations as Propagator
- Specialized quantum control methods: CRAB, GROUP, GOAT, etc.
- But: local traps, controllability issues