



Optimal Control Techniques for Quantum Interferometry

Michael H. Goerz

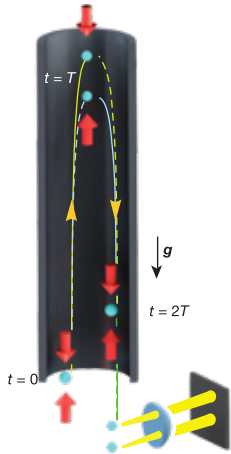
DEVCOM Army Research Lab

IMSI Quantum Hardware Workshop, Chicago, October 30, 2024

Quantum Interferometry

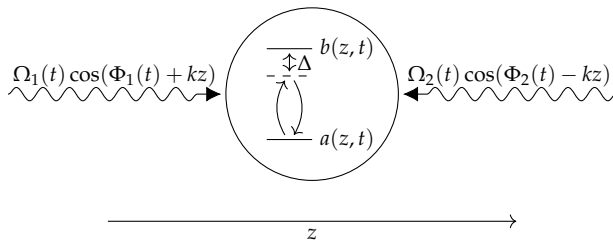
- Ensemble of atoms
- Create a superposition
- Separate components
- Accumulate a relative phase
- Recombine
- Imprint phase on measurement

Atomic Fountain Interferometer

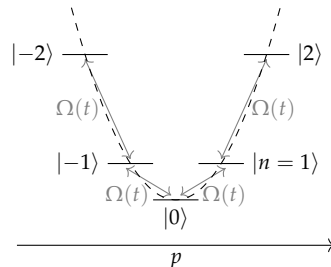


Atomic Fountain Interferometer

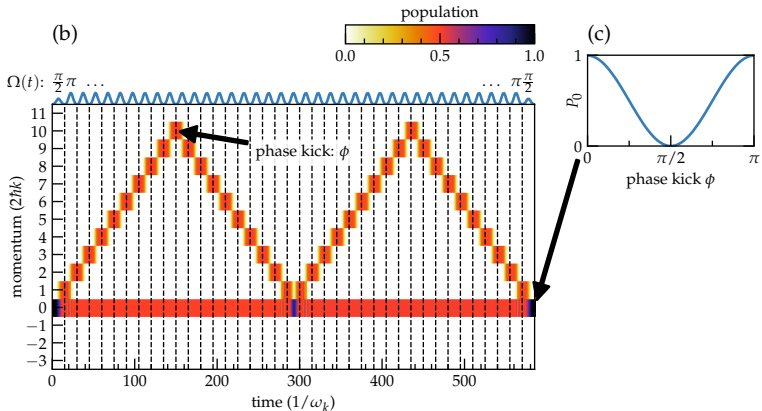
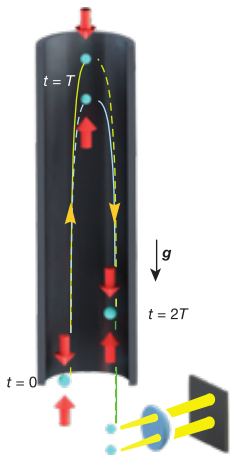
(a)



(b)

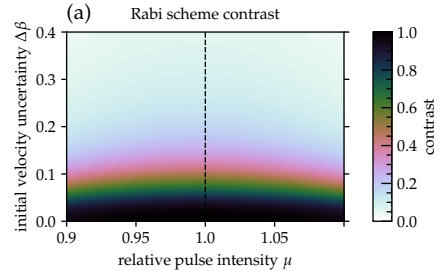
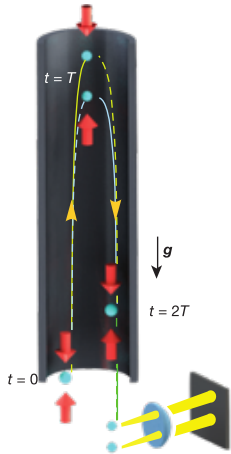


— Goerz, Kasevich, Malinovsky. Atoms 11, 36 (2023)



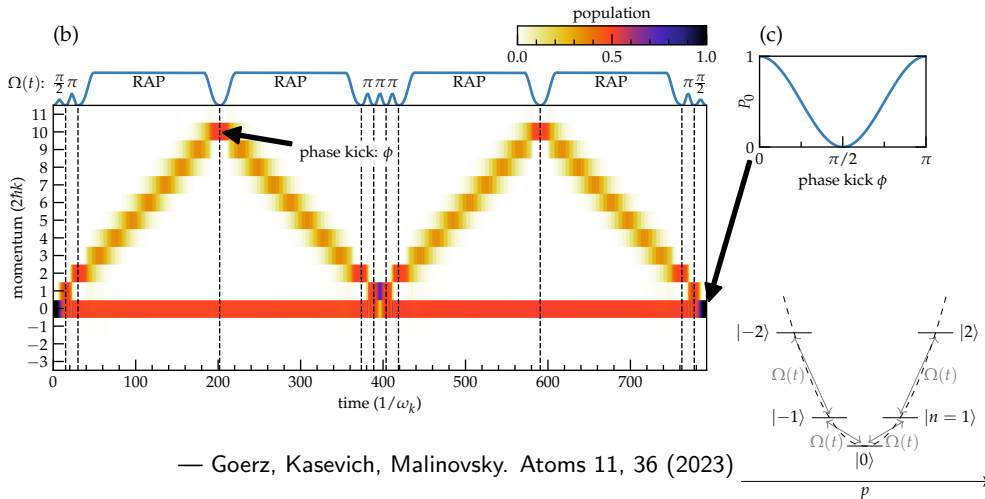
— Goerz, Kasevich, Malinovsky. *Atoms* 11, 36 (2023)

Atomic Fountain Interferometer – Robustness

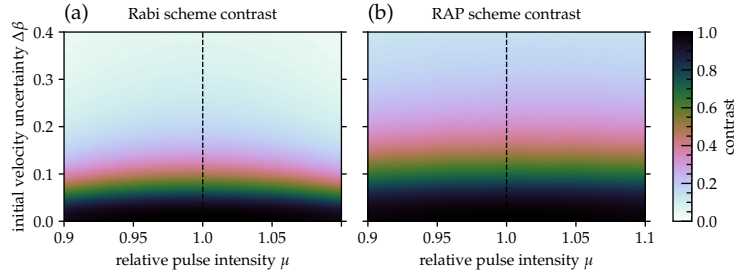
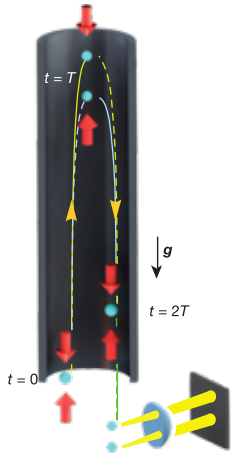


— Goerz, Kasevich, Malinovsky. *Atoms* 11, 36 (2023)

Atomic Fountain Interferometer – Rapid Adiabatic Passage (RAP)

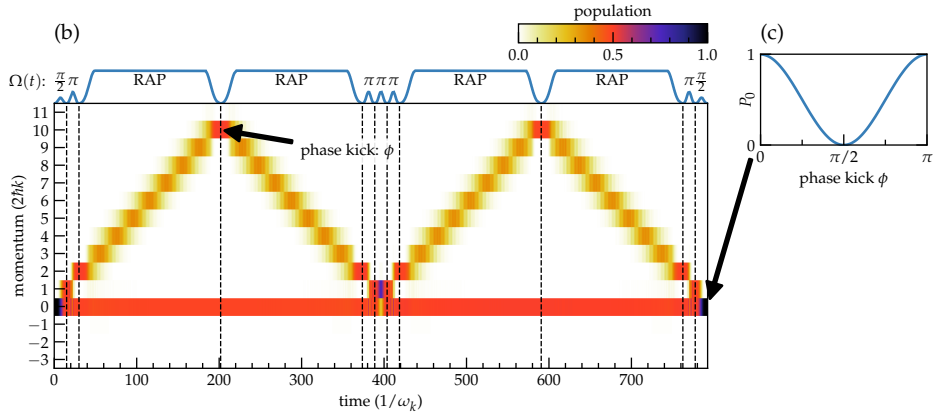


Atomic Fountain Interferometer – Robustness RAP



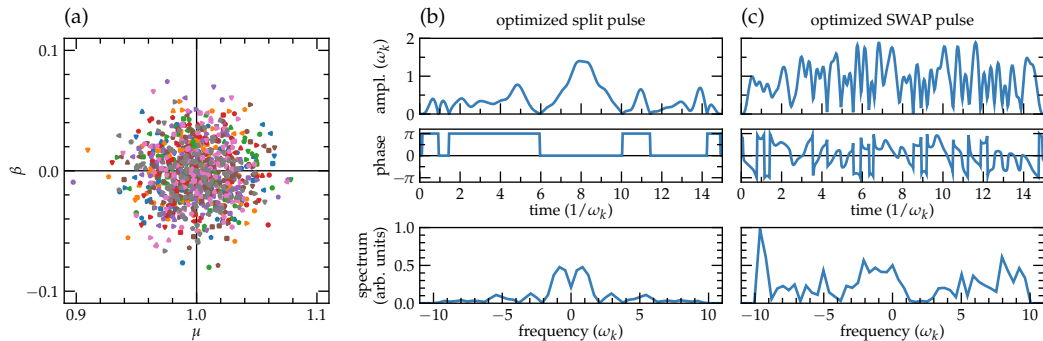
— Goerz, Kasevich, Malinovsky. *Atoms* 11, 36 (2023)

Atomic Fountain Interferometer – Optimal Control (OCT)



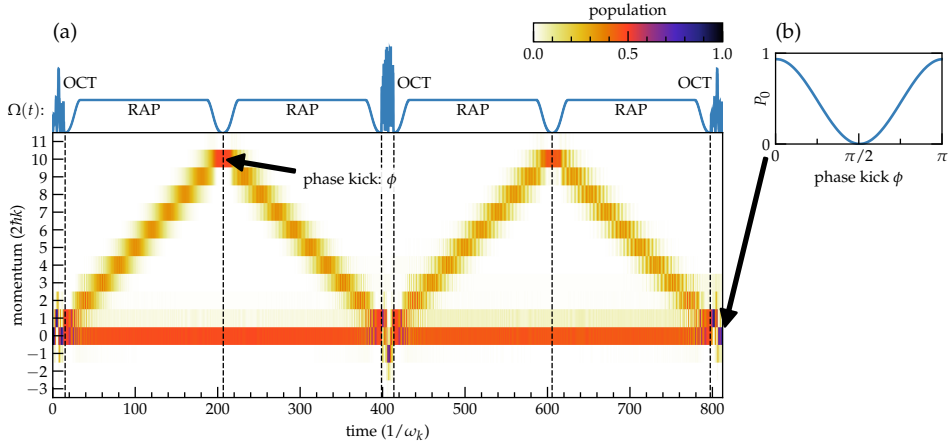
— Goerz, Kasevich, Malinovsky. Atoms 11, 36 (2023)

Atomic Fountain Interferometer – Optimal Control (OCT)



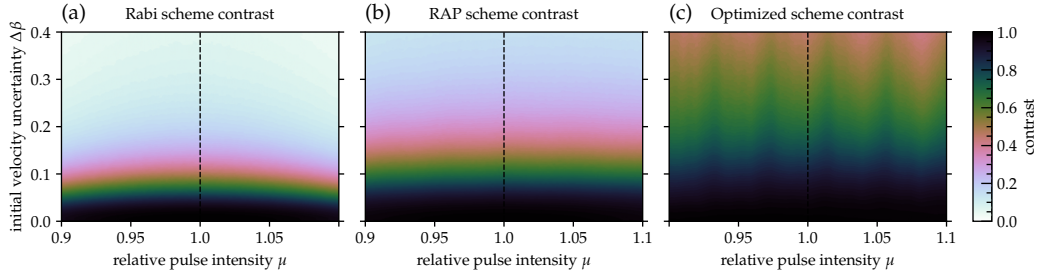
— Goerz, Kasevich, Malinovsky. *Atoms* 11, 36 (2023)

Atomic Fountain Interferometer – Optimal Control (OCT)



— Goerz, Kasevich, Malinovsky. *Atoms* 11, 36 (2023)

Atomic Fountain Interferometer – Robustness OCT



— Goerz, Kasevich, Malinovsky. *Atoms* 11, 36 (2023)

JuliaQuantumControl



JuliaQuantumControl

Julia Framework for Quantum Optimal Control

42 followers <https://juliaquantumcontrol.github.io...>

Overview Repositories 18 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.

The packages in [JuliaQuantumControl](#) that implement specific individual methods should be used via the high-level package

People



Top languages

Julia TeX Jupyter Notebook

Most used topics

julia quantum grape
optimal-control quantum-computing

Why Julia?

- Performance – compiles to low-level machine code (matches Fortran)
- Flexibility – multiple dispatch, ecosystem
- Expressiveness – clean syntax, unicode, notebook environment

JuliaQuantumControl



JuliaQuantumControl

Julia Framework for Quantum Optimal Control

42 followers [https://juliaquantumcontrol.github.io/...](https://juliaquantumcontrol.github.io/)

[Overview](#) [Repositories 18](#) [Discussions](#) [Projects](#) [Packages](#) [People 3](#)

README.md

A Julia Framework for Quantum Optimal Control.

[docs](#) [stable](#) [docs](#) [dev](#)

People



Top languages

Design Principle

Maximum performance and composability through abstract interfaces

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.

The packages in [JuliaQuantumControl](#) that implement specific individual methods should be used via the high-level package

[julia](#) [quantum](#) [grape](#)
[optimal-control](#) [quantum-computing](#)

Optimization Functional

QuantumControl.jl

 JuliaQuantumControl
  QuantumControl.jl
  version
  v0.11.1+dev

QuantumControl.jl is a [Julia framework for quantum dynamics and control](#).

Quantum optimal control [1–8] 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 ("pulse level control"). 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](#) [9–14] and [GRAPE](#) [15, 16] 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.

Functional — Mathematically, the control problem is the minimization of a functional of the form

$$J(\{\epsilon_l(t)\}) = J_T(\{|\Psi_k(T)\rangle\}) + \underbrace{\lambda_a \sum_l \int_0^T g_a(\epsilon_l(t)) dt}_{=J_a(\{\epsilon_l(t)\})} + \underbrace{\lambda_b \sum_k \int_0^T g_b(|\Psi_k(t)\rangle) dt}_{=J_b(\{|\Psi_k(t)\rangle\})},$$

Control Problem and Trajectories

✓ `QuantumControl.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.

Control Problem and Trajectories

QuantumControl.Trajectory – Type

Description of a state's time evolution.

```
Trajectory(  
    initial_state,  
    generator;  
    target_state=nothing,  
    weight=1.0,  
    kwargs...  
)
```

describes the time evolution of the `initial_state` under a time-dependent dynamical `generator` (e.g., a Hamiltonian or Liouvillian).

Trajectories are central to quantum control problems: an optimization functional depends on the result of propagating one or more trajectories. For example, when optimizing for a quantum gate, the optimization considers the trajectories of all logical basis states.

In addition to the `initial_state` and `generator`, a `Trajectory` may include data relevant to the propagation and to evaluating a particular optimization functional. Most functionals have the notion of a

Dynamical Generators



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,

$$\hat{H} = \overbrace{\hat{H}_0}^{\text{drift term}} + \sum_l \overbrace{\hat{H}_l(\{\epsilon_l(t)\}, t)}^{\text{control term}} \quad (\text{G1})$$

$$\hat{H} = \hat{H}_0 + \sum_l \overbrace{a_l(\{\epsilon_l(t)\}, t)}^{\text{control amplitude}} \underbrace{\hat{H}_l}_{\text{control function}} \quad (\text{G2})$$

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

In Eq. (G1), $\hat{H}_0$ is the **Drift Term** (which may be zero) and each term under the sum over l is a **Control Term**. Each

Dynamical Generators

✓ `QuantumPropagators.Interfaces.check_generator` — Function

Check the dynamical generator for propagating state over tlist.

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

verifies the given generator:

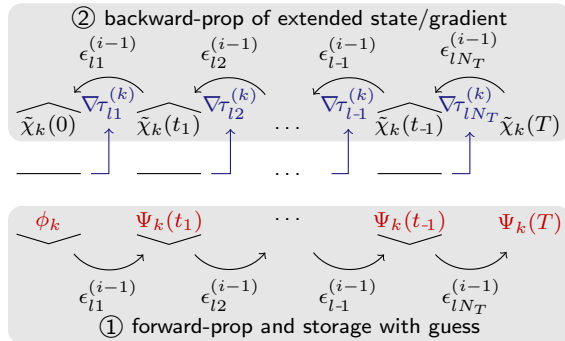
- `get_controls(generator)` must be defined and return a tuple
- all controls returned by `get_controls(generator)` must pass `check_control`
- `substitute(generator, replacements)` must be defined
- If `generator` is a `Generator` instance, all elements of `generator.amplitudes` must pass `check_amplitude`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

If `for_pwc` (default):

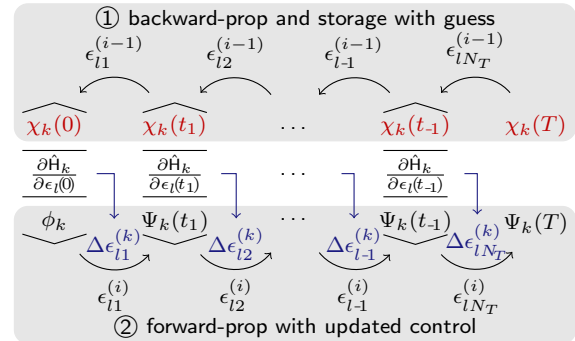
- `op = evaluate(generator, tlist, n)` must return a valid operator (`check_operator`), with forwarded

Optimization schemes

(a) GRAPE



(b) Krotov's method



— Goerz, Carrasco, Malinovsky. Quantum 6, 871 (2022)

Propagators

✓ `QuantumPropagators.AbstractPropagator` — Type

Abstract base type for all `Propagator` objects.

All `Propagator` objects must be instantiated via `init_prop` and implement the following interface.

Properties

- `state` (read-only): The current quantum state in the propagation
- `tlist` (read-only): The time grid for the propagation
- `t` (read-only): The time at which `state` is defined. An element of `tlist`.
- `parameters`: parameters that determine the dynamics. The structure of the parameters depends on the concrete `Propagator` type (i.e., the propagation method). Mutating the `parameters` affects subsequent propagation steps.
- `backward`: Boolean flag to indicate whether the propagation moves forward or backward in time
- `inplace`: Boolean flag to indicate whether `propagator.state` is modified in-place or is recreated by every call of `prop_step!` or `set_state!`. With `inplace=false`, the propagator should generally avoid in-place operations, such as calls to `QuantumPropagators.Controls.evaluate!`.

Concrete `Propagator` types may have additional properties or fields, but these should be considered private.

Propagators

every call of `prop_step!` or `set_state!`. With `inplace=false`, the propagator should generally avoid in-place operations, such as calls to `QuantumPropagators.Controls.evaluate!`.

Concrete `Propagator` types may have additional properties or fields, but these should be considered private.

Methods

- `reinit_prop!` — reset the propagator to a new initial state at the beginning of the time grid (or the end, for backward propagation)
- `prop_step!` — advance the propagator by one step forward (or backward) on the time grid.
- `set_state!` — safely mutate the current quantum state of the propagation. Note that directly mutating the state property is not safe. However, $\Psi = \text{propagator.state}$; `foo_mutate!( $\Psi$ )`, `set_state!(propagator,  $\Psi$ )` for some mutating function `foo_mutate!` is guaranteed to be safe and efficient for both in-place and not-in-place propagators.
- `set_t!` — safely mutate the current time (`propagator.t`), snapping to the values of `tlist`.

See also

- `PiecewisePropagator` — specialization of `AbstractPropagator` for piecewise propagation methods.
- `PWCPropagator` — specialization of `PiecewisePropagator` for piecewise-constant propagation methods. [source](#)

Propagators


[HOME](#)
[MODELING ▾](#)
[SOLVERS ▾](#)
[ANALYSIS ▾](#)
[MACHINE LEARNING ▾](#)
[DEVELOPER TOOLS ▾](#)



DifferentialEquations.jl

DifferentialEquations.jl: Efficient Differential Equation Solving in Julia

- Contributing
- Supporting and Citing
- Getting Started: Installation And First Steps
- Acknowledgements
- Reproducibility

Getting Started with Differential Equations in Julia

Version	v7.14.0	▾
---------	---------	---

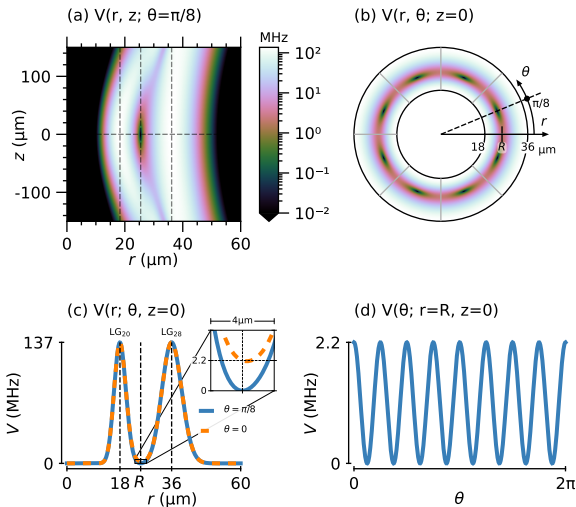
DifferentialEquations.jl: Efficient Differential Equation Solving in Julia

[GitHub](#) [✎](#) [⚙](#) [^](#)

This is a suite for numerically solving differential equations written in Julia and available for use in Julia, Python, and R. The purpose of this package is to supply efficient Julia implementations of solvers for various differential equations. Equations within the realm of this package include:

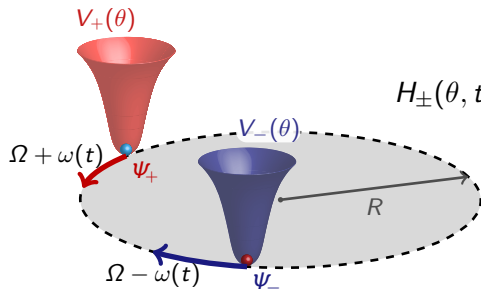
- Discrete equations (function maps, discrete stochastic (Gillespie/Markov) simulations)
- Ordinary differential equations (ODEs)
- Split and Partitioned ODEs (Symplectic integrators, IMEX Methods)
- Stochastic ordinary differential equations (SODEs or SDEs)
- Stochastic differential-algebraic equations (SDAEs)
- Random differential equations (RODEs or RDEs)
- Differential algebraic equations (DAEs)
- Delay differential equations (DDEs)
- Neutral, retarded, and algebraic delay differential equations (NDDEs, RDDEs, and DDAEs)
- Stochastic delay differential equations (SDDEs)
- Experimental support for stochastic neutral, retarded, and algebraic delay differential equations (SNDDs)

Rotating Tractor Interferometer



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

Rotating Tractor Interferometer



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

In co-moving frame:

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

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

Project-Specific Data Structures

Rotating TAI Implementation

```

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

```

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

```

function LinearAlgebra.mul!(C, A::SplitOperator, B, α, β)
    #  $|C\rangle = \beta |C\rangle + \alpha \hat{A} |B\rangle = (\beta |C\rangle + \alpha \hat{V} |B\rangle) + \alpha \hat{T} |B\rangle$ 
    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

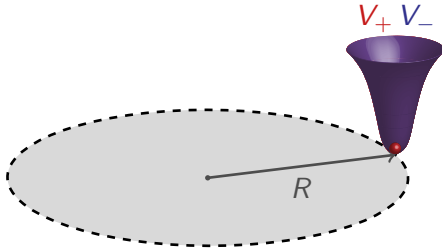
```

N 8% 38/444: 1 "α rotating_tai.jl

<julia<master

Adiabatic Dynamics of Rotating TAI

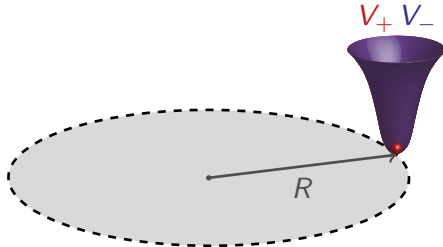
$t = 0$ ms



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

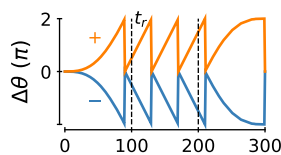
Adiabatic Dynamics of Rotating TAI

$t = 300$ ms

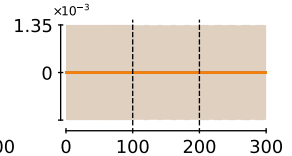


$$\omega(t) = \begin{cases} \omega_0 \sin^2\left(\frac{\pi t}{2t_r}\right) & 0 \leq t < t_r \\ \omega_0 & t_r \leq t < t_r + t_{\text{loop}} \\ \omega_0 \cos^2\left(\frac{\pi t'}{2t_r}\right) & T - t_r \leq t \leq T \end{cases}$$

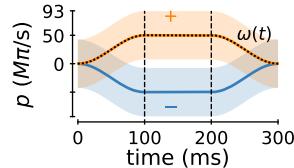
lab frame position



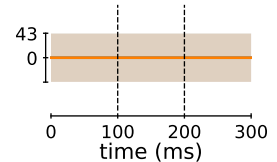
moving frame position



lab frame momentum



moving frame momentum



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

Interferometric Response of Rotating TAI

$$\Delta\Phi_S = \frac{4m\Omega A}{\hbar}, \quad A = \frac{R^2}{2} \underbrace{\int_0^T \omega(t') dt'}_{=n\pi}$$

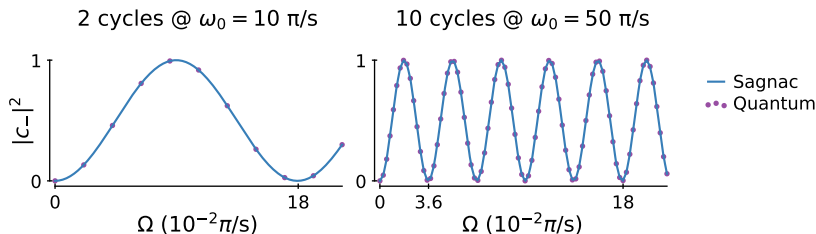
$$|c_{\pm}|^2 = \frac{1}{2} \pm \frac{1}{2} \text{Re} \left[\eta e^{-i\Delta\Phi} \right] \quad \rightarrow \quad |c_-|^2 = \frac{1}{2} - \frac{\cos \Delta\Phi}{2} = \sin^2 \left(\frac{\Delta\Phi}{2} \right)$$

$$\eta = \langle \Psi_-(\theta, T) | \Psi_+(\theta, T) \rangle = 1 \quad \text{if adiabatic}$$

Interferometric Response of Rotating TAI

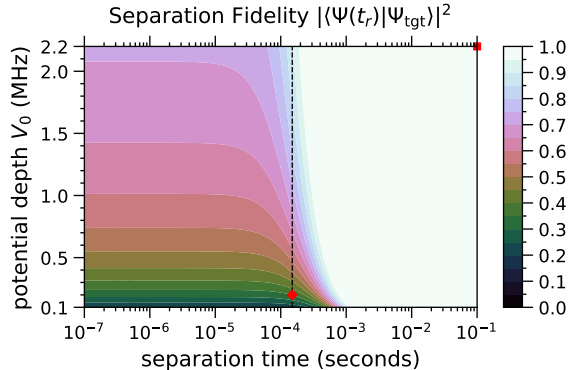
$$\Delta\Phi_S = \frac{4m\Omega A}{\hbar}, \quad A = \frac{R^2}{2} \underbrace{\int_0^T \omega(t') dt'}_{=10\pi}$$

$$|c_{\pm}|^2 = \frac{1}{2} \pm \frac{1}{2} \text{Re} \left[\eta e^{-i\Delta\Phi} \right] \quad \rightarrow \quad |c_-|^2 = \frac{1}{2} - \frac{\cos \Delta\Phi}{2} = \sin^2 \left(\frac{\Delta\Phi}{2} \right)$$



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

Non-Adiabatic Dynamics of Rotating TAI

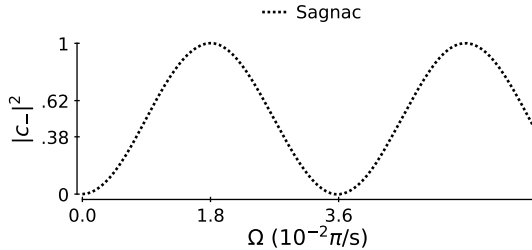


$$\omega(t) = \begin{cases} \omega_0 \sin^2 \left(\frac{\pi t}{2t_r} \right) & 0 \leq t < t_r \\ \omega_0 & t_r \leq t < t_r + t_{\text{loop}} \\ \omega_0 \cos^2 \left(\frac{\pi t'}{2t_r} \right) & T - t_r \leq t \leq T \end{cases}$$

$|\Psi_{\text{tgt}}\rangle = \text{ground state of moving potential}$

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

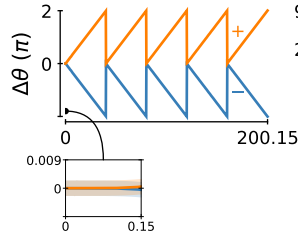
Non-Adiabatic Dynamics of Rotating TAI



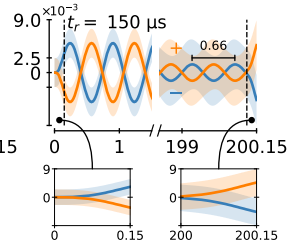
$$\Delta\Phi_S = \frac{4m\Omega A}{\hbar}, \quad A = \frac{R^2}{2} \cdot 10\pi$$

$$|c_-|^2 = \frac{1}{2} - \frac{\cos \Delta\Phi}{2} = \sin^2 \left(\frac{\Delta\Phi}{2} \right)$$

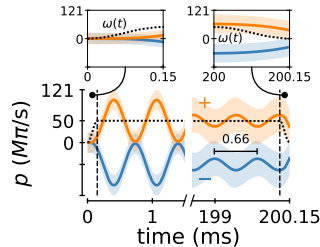
lab frame position



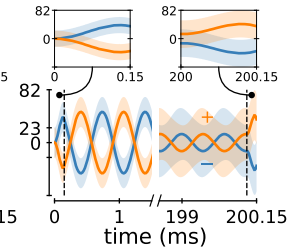
moving frame position



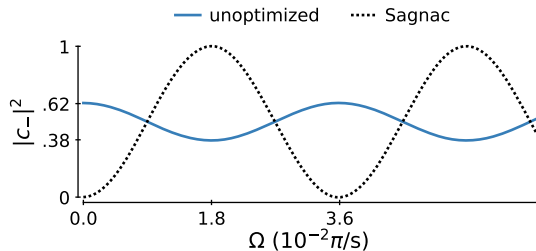
lab frame momentum



moving frame momentum



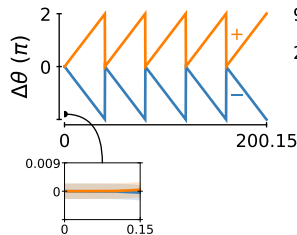
Non-Adiabatic Dynamics of Rotating TAI



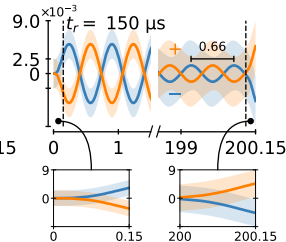
$$\Delta\Phi_S = \frac{4m\Omega A}{\hbar}, \quad A = \frac{R^2}{2} \cdot 10\pi$$

$$|c_-|^2 = \frac{1}{2} - \frac{1}{2}\text{Re} \left[\eta e^{-i\Delta\Phi} \right]$$

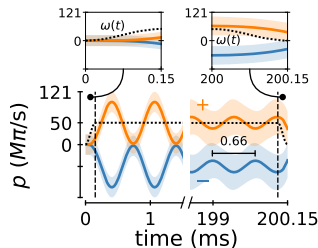
lab frame position



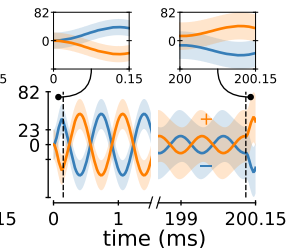
moving frame position



lab frame momentum



moving frame momentum



Rotating TAI Control Problem

$$\omega(t) = \begin{cases} \omega_{\text{opt}}(t) & 0 \leq t < t_r \\ \omega_0 & t_r \leq t < t_r + t_{\text{loop}} \\ \omega_{\text{opt}}(t') & T - t_r \leq t \leq T \end{cases}$$

Find $\omega_{\text{opt}}(t)$ for short t_r so that

$$\Psi(\theta, t = 0) \rightarrow \Psi_{\text{tgt}}(\theta, t = t_r)$$

where $|\Psi_{\text{tgt}}\rangle =$ ground state of moving potential

Optimization with QuantumControl.jl

2023-05-17_OCT_tr=150 μ s_V0=0.2MHz_R=26 μ m_ ω =50 π ps.ipynb

File Edit View Run Kernel Tabs Settings Help

git

Interface Julia 1.8.5

```
[38]: objective = Objective(
        initial_state=INITIAL_STATE,
        target_state=TARGET_STATE,
        generator=set_guided_control(HAMILTONIAN, TIME_GRID)
    );
     $\delta\omega$  = get_controls(objective.generator)[1];
```

Last executed at 2023-06-04 15:27:22 in 711ms

```
[39]: problem = ControlProblem(;
        objectives=[objective], tlist=TIME_GRID,
        J_T=J_T_sm, prop_method=:splitprop, verbose=false,
        pulse_options=IdDict( $\delta\omega \Rightarrow$  Dict(:lambda_a  $\Rightarrow$  1e6, :update_shape  $\Rightarrow$  t  $\rightarrow$  1.0)),
        check_convergence=res  $\rightarrow$  begin ((res.J_T < 1e-8) && (res.converged = true) && (res.message = "J_T < 10-8"))
    );
```

Last executed at 2023-06-04 15:27:22 in 284ms

```
[40]: res = @optimize_or_load("./data/$NAME.jld2", problem; method=:krotov, iter_stop=400, force=true)
```

Last executed at 2023-06-04 15:28:04 in 42.53s

iter.	J_T	$\int g_a(t)dt$	J	ΔJ_T	ΔJ	secs
0	6.48e-01	0.00e+00	6.48e-01	n/a	n/a	0.9
1	6.38e-01	4.99e-03	6.43e-01	-1.01e-02	-5.09e-03	5.3
2	6.27e-01	5.13e-03	6.33e-01	-1.04e-02	-5.23e-03	0.1
3	6.17e-01	5.28e-03	6.22e-01	-1.07e-02	-5.38e-03	0.1
4	6.06e-01	5.42e-03	6.11e-01	-1.09e-02	-5.53e-03	0.1

Optimization with QuantumControl.jl



2023-05-17_OCT_tr=150μs_V0=0.2MHz_R=26μm_ω=50πps.ipynb

File Edit View Run Kernel Tabs Settings Help

Interface Julia 1.8.5

```
[38]: objective = Objective(
        initial_state=INITIAL_STATE,
        target_state=TARGET_STATE,
        generator=set_guided_control(HAMILTONIAN, TIME_GRID)
    );
    δω = get_controls(objective.generator)[1];
```

Last executed at 2023-06-04 15:27:22 in 711ms

Guided Control

[39]:

$$\omega_{\text{opt}}(t) = \omega(t) + S(t)\delta\omega(t)$$

```
[40]: res = @optimize_or_load("./data/$NAME.jld2", problem; method=:krotov, iter_stop=400, force=true)
```

Last executed at 2023-06-04 15:28:04 in 42.53s

iter.	J_T	/g _a (t)dt	J	ΔJ_T	ΔJ	secs
0	6.48e-01	0.00e+00	6.48e-01	n/a	n/a	0.9
1	6.38e-01	4.99e-03	6.43e-01	-1.01e-02	-5.09e-03	5.3
2	6.27e-01	5.13e-03	6.33e-01	-1.04e-02	-5.23e-03	0.1
3	6.17e-01	5.28e-03	6.22e-01	-1.07e-02	-5.38e-03	0.1
4	6.06e-01	5.42e-03	6.11e-01	-1.09e-02	-5.53e-03	0.1

Optimization with QuantumControl.jl

2023-05-17_OCT_tr=150 μ s_V0=0.2MHz_R=26 μ m_ ω =50 π ps.ipynb

File Edit View Run Kernel Tabs Settings Help

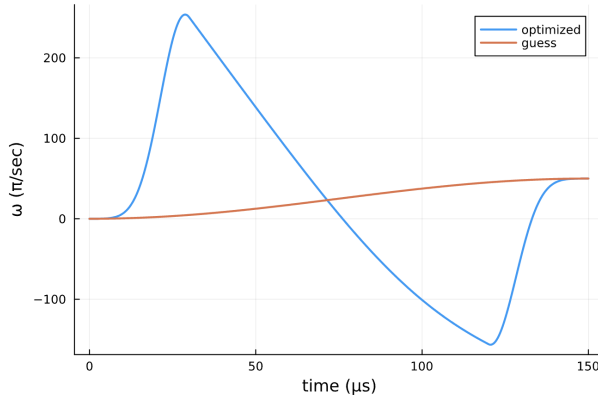
+ ✂ 📄 📌 ▶ ⏮ ⏭ Markdown ⌵ ⌚ git

Interface ⚙ Julia 1.8.5 ○

```
plot!(; xlabel="time ( $\mu$ s)", ylabel=" $\omega$  ( $\pi$ /sec)")
```

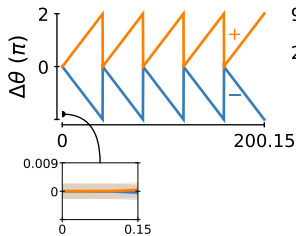
Last executed at 2023-06-06 19:25:30 in 437ms

[45]:

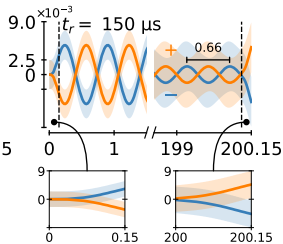


Optimized Dynamics or Rotating TAI

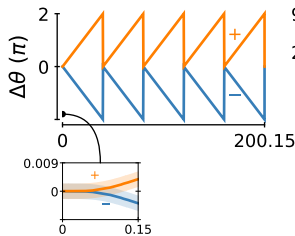
lab frame position



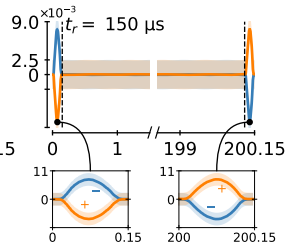
moving frame position



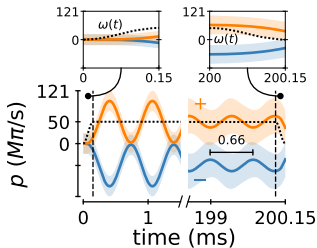
lab frame position



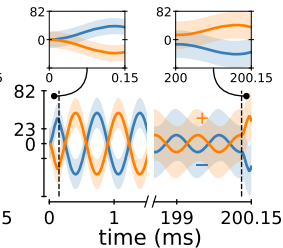
moving frame position



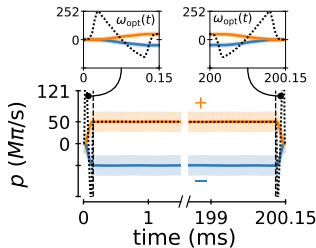
lab frame momentum



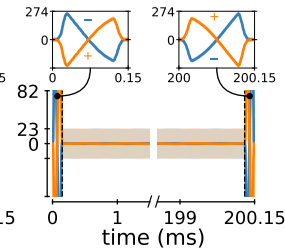
moving frame momentum



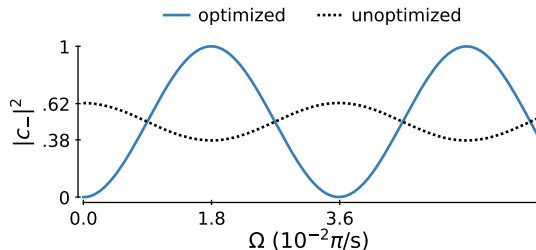
lab frame momentum



moving frame momentum



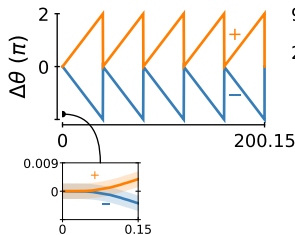
Optimized Dynamics or Rotating TAI



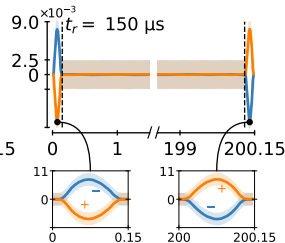
$$\Delta\Phi_S = \frac{4m\Omega A}{\hbar}, \quad A = \frac{R^2}{2} \cdot 10\pi$$

$$|c_-|^2 = \frac{1}{2} - \frac{\cos \Delta\Phi}{2} = \sin^2\left(\frac{\Delta\Phi}{2}\right)$$

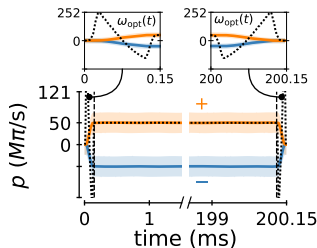
lab frame position



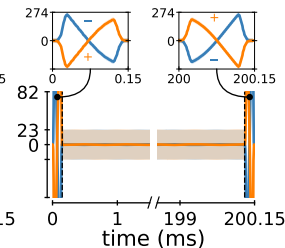
moving frame position



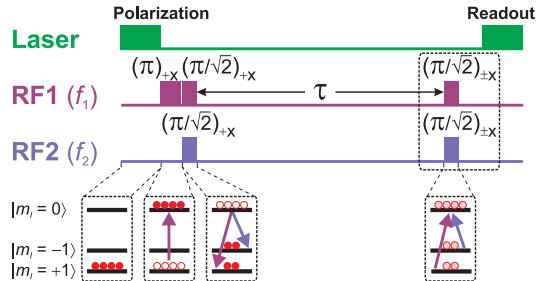
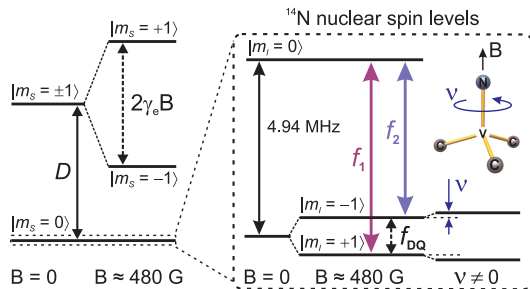
lab frame momentum



moving frame momentum

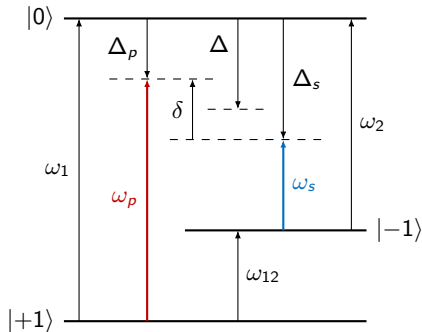


Nuclear Spin Gyroscope



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

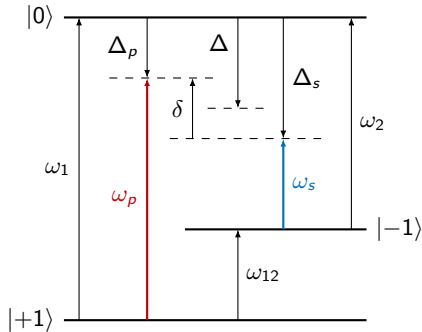
NV Center Hamiltonian with Crosstalk



$$\hat{H} = \begin{pmatrix} 0 & \Omega(t) & 0 \\ \Omega(t) & \omega_1 & \Omega(t) \\ 0 & \Omega(t) & \omega_1 - \omega_2 \end{pmatrix}$$

$$\begin{aligned} \Omega(t) = & \Omega_p(t) \cos(\omega_p t + \phi_p(t) + \phi_p^{(i)}) \\ & + \Omega_s(t) \cos(\omega_s t + \phi_s(t) + \phi_s^{(i)}) \end{aligned}$$

NV Center Hamiltonian with Crosstalk



$$\hat{H}_{\text{RWA}} = \begin{pmatrix} -\frac{\delta}{2} & \Omega_{1,0}(t) & 0 \\ \Omega_{1,0}^*(t) & \Delta & \Omega_{0,-1}(t) \\ 0 & \Omega_{0,-1}^*(t) & \frac{\delta}{2} \end{pmatrix}$$

$$\Omega_{1,0}(t) = \frac{\Omega_p(t)}{2} e^{i\phi_p^{(i)}} + \frac{\Omega_s(t)}{2} e^{i\phi_s^{(i)}} e^{-i\omega_{ps}t}$$

$$\Omega_{0,-1}(t) = \frac{\Omega_s^*(t)}{2} e^{-i\phi_p^{(i)}} + \frac{\Omega_p^*(t)}{2} e^{-i\phi_s^{(i)}} e^{-i\omega_{ps}t}$$

NV Center Hamiltonian with Crosstalk

```
@kwdef struct RamseyHamiltonian
```

```
    "Single-photon detuning"
```

```
    Δ::Float64 = 0.0
```

```
    "Two-Photon detuning"
```

```
    δ::Float64 = 0.0
```

```
    "Scaling factor for field envelopes"
```

```
    μ::Float64 = 1.0
```

```
    "Time before free time evolution"
```

```
    T₁::Float64
```

```
    """The frequency ``ω_{ps} = ω_p - ω_s`` at which the crosstalk
    the Hamiltonian oscillate. If zero (default), no cross-talk
    account"""
```

```
    ω_{ps}::Float64 = 0.0
```

```
    "Phase offset of the pump field before the free time evolution"
```

```
    φ_{p1}::Float64 = 0.0
```

```
    "Phase offset of the pump field after the free time evolution"
```

```
    φ_{p2}::Float64 = 0.0
```

```
    "Phase offset of the Stokes field before the free time evolution"
```

```
    φ_{s1}::Float64 = 0.0
```

```
    "Phase offset of the Stokes field after the free time evolution"
```

```
    φ_{s2}::Float64 = 0.0
```

```
function _evaluate(
```

```
    H::RamseyHamiltonian,
```

```
    t::Float64,
```

```
    Ω_{sre}::Float64,
```

```
    Ω_{sim}::Float64,
```

```
    Ω_{pre}::Float64,
```

```
    Ω_{pim}::Float64,
```

```
)
```

```
@unpack Δ, δ, μ, T₁, ω_{ps}, φ_{p1}, φ_{p2}, φ_{s1}, φ_{s2} = H
```

```
    Φ₁ = (t ≤ T₁) ? exp(i * φ_{p1}) : exp(i * φ_{p2})
```

```
    Φ₂ = (t ≤ T₁) ? exp(i * φ_{s1}) : exp(i * φ_{s2})
```

```
    Ω₁ = (Ω_{pre} + i * Ω_{pim})
```

```
    Ω₂ = (Ω_{sre} + i * Ω_{sim})
```

```
    Ω_{10} = (μ / 2) * Ω₁ * Φ₁
```

```
    Ω_{0-1} = (μ / 2) * Ω₂' * Φ₂' # note the complex conjugate
```

```
    if ω_{ps} ≠ 0.0
```

```
        # fast-oscillating cross-talk terms
```

```
        Ω_{10} += (μ / 2) * Ω₂ * Φ₂ * exp(-i * ω_{ps} * t)
```

```
        Ω_{0-1} += (μ / 2) * Ω₁' * Φ₁' * exp(-i * ω_{ps} * t)
```

```
    end
```

```
    #!format: off
```

```
    return @SMatrix{ComplexF64,
```

```
        -δ/2      Ω_{10}      0
        Ω_{10}'    Δ          Ω_{0-1}
        0          Ω_{0-1}'    δ/2
```

```
]
```

```
    #!format: on
```

Parameterized Pulses



2024-09-17_SpectrumOCT_PhaseControl_TanhParam.ipynb

File Edit View Run Kernel Tabs Settings Help

Code

Julia 1.10.3

IP

```
[19]: Qpre_guess = ParameterizedAmplitude(
      t → shape_p_guess(t), tlist, shape = update_shape,
      parameterization=TanhParameterization(-30kHz, 30kHz),
      parameterize=true
    );
    Qpim_guess = ParameterizedAmplitude(
      t → 0.0, tlist, shape = update_shape,
      parameterization=TanhParameterization(-30kHz, 30kHz),
      parameterize=true
    );
    Qsre_guess = ParameterizedAmplitude(
      t → shape_s_guess(t), tlist, shape = update_shape,
      parameterization=TanhParameterization(-30kHz, 30kHz),
      parameterize=true
    );
    Qsim_guess = ParameterizedAmplitude(
      t → 0.0, tlist, shape = update_shape,
      parameterization=TanhParameterization(-30kHz, 30kHz),
      parameterize=true
    );
```

Last executed at 2024-10-28 07:56:00 in 301ms

```
[20]: plot(tlist ./ μs, discretize(Array(Qpre_guess), tlist) ./ kHz; label = "Qp")
      plot!(tlist ./ μs, discretize(Array(Qsre_guess), tlist) ./ kHz; label = "Qs")
      vline!([T, 2T] ./ μs; color = "black", ls = :dash, label = "")
      plot!(); xlabel = "time (μs)", ylabel = "amplitude (kHz)"
```

Last executed at 2024-10-28 07:56:02 in 1.61s

Simple 1 5 Julia 1.10.3 | Idle

Mode: Edit Ln 15, Col 3 2024-09-17_SpectrumOCT_PhaseControl_TanhParam.ipynb



Parameterized Pulses

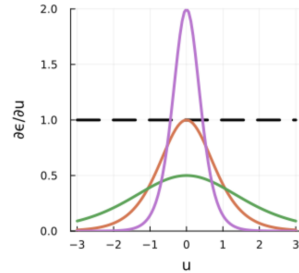
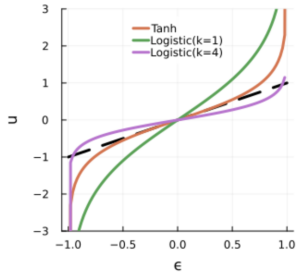
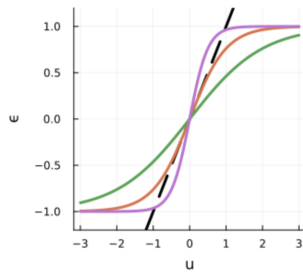


Tutorials / Pulse Parametrization for Krotov's Method

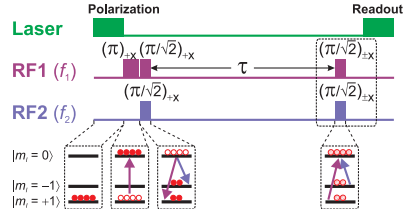
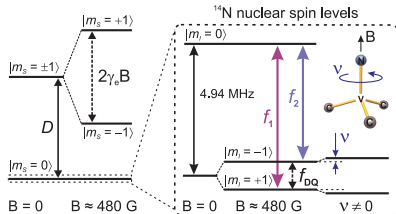
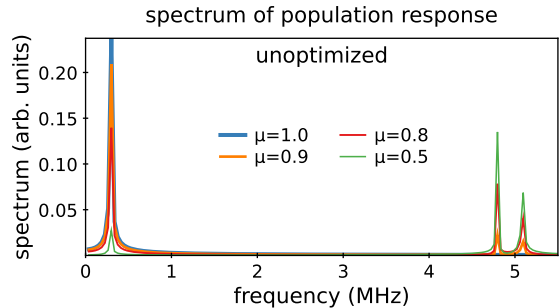
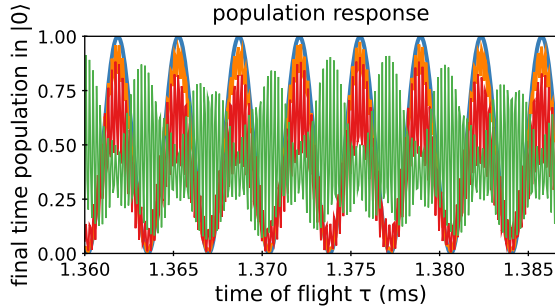


Parametrizations

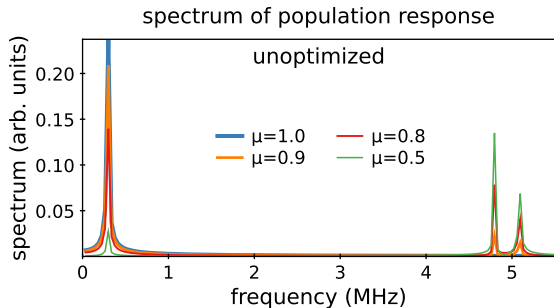
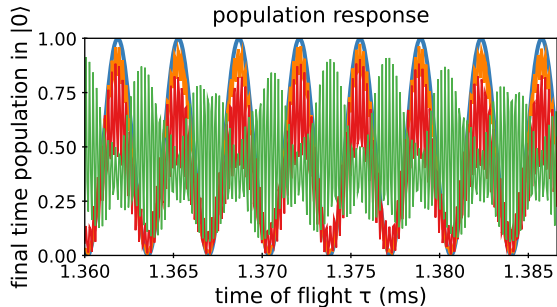
Symmetric Bounded Controls



Population Response Signal



Population Response Signal



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

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

NV Center Optimization Problem

2024-09-10_FullSchemePropagationWithCallback.ipynb

File Edit View Run Kernel Tabs Settings Help

Code git

Julia 1.10.3

```

trajectories = vec([
    begin
        H = ham(; tlist, μ, Q_pre, Q_pim, Q_sre, Q_sim, φ_p2=(ω_p*τ), φ_s2=(ω_s*τ), crosstalk)
        if use_prop_callback
            prop_callback = FreeTimeEvolutionCallback(H; τ)
            Trajectory(Ψ_0, H; prop_callback)
        else
            Trajectory(Ψ_0, H)
        end
    end
    for (τ, μ) in Iterators.product(τ_vals, μ_vals)
end
])

if _return == :trajectories
    return trajectories
end

function J_T_spectrum(Ψ, trajectories; kwargs...)
end

```

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

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

Simple 1 5 Julia 1.10.3 | Idle Mode: Command Ln 1, Col 1 2024-09-10_FullSchemePropagationWithCallback.ipynb 0

NV Center Optimization Problem



2024-09-10_FullSchemePropagationWithCallback.ipynb

File Edit View Run Kernel Tabs Settings Help

Code git

Julia 1.10.3

```

trajectories = vec([
    begin
        H = ham(; tlist, μ, Ωpre, Ωpim, Ωsre, Ωsim, φp2=(ωp*τ), φs2=(ωs*τ) crosstalk)
        if use_prop_callback
            prop_callback = FreeTimeEvolutionCallback(H; τ)
            Trajectory(Ψ₀, H; prop_callback)
        else
            Trajectory(Ψ₀, H)
        end
    end
    for (τ, μ) in Iterators.product(τ_vals, μ_vals)
])

if _return == :trajectories
    return trajectories
end

function J_T_spectrum(Ψ, trajectories; kwargs...)

```

$$\Omega_{1,0}(t) = \frac{\Omega_p(t)}{2} e^{i\phi_p^{(i)}} + \frac{\Omega_s(t)}{2} e^{i\phi_s^{(i)}} e^{-i\omega_{ps}t}$$

Absorb phase difference in RF pulse

```

return norm(spec - spect_target)
end

```

Simple 1 5 Julia 1.10.3 | Idle

Mode: Command Ln 1, Col 1 2024-09-10_FullSchemePropagationWithCallback.ipynb 0

NV Center Optimization Problem



2024-09-10_FullSchemePropagationWithCallback.ipynb

File Edit View Run Kernel Tabs Settings Help

Code git

Julia 1.10.3

```

trajectories = vec([
    begin
        H = ham(; tlist, μ, Q_pre, Q_pim, Q_sre, Q_sim, φ_p2=(ω_p*τ), φ_s2=(ω_s*τ), crosstalk)
        if
            prop_callback = FreeTimeEvolutionCallback(H; τ)
            Trajectory(Ψ_0, H; prop_callback)
        else
            Trajectory(Ψ_0, H)
        end
    end
    for (τ, μ) in Iterators.product(τ_vals, μ_vals)
])

if _return == :trajectories
    return trajectories
end

function J_T_spectrum(Ψ, trajectories; kwargs...)

```

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

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

Simple 1 5 Julia 1.10.3 | Idle

Mode: Command Ln 1, Col 1 2024-09-10_FullSchemePropagationWithCallback.ipynb 0

NV Center Optimization Problem



2024-09-10_FullSchemePropagationWithCallback.ipynb

File Edit View Run Kernel Tabs Settings Help

Code git

Julia 1.10.3

```
trajectories = vec([
    begin
```

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

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

```
)
```

```
if _return == :trajectories
    return trajectories
end
```

```
function J_T_spectrum(Ψ, trajectories; kwargs...)
    M = length(μ_vals)
    N = length(τ_vals)
    spec = zeros(N)
    for m = 1:M
        states = collect(Ψ[((m-1)*N+1):m*N])
        pop0 = [abs2(Ψ[2]) for Ψ in states]
        spec = spec .+ (abs2.(FFTW.fft(pop0)) ./ M)
    end
    return norm(spec - spect_target)
end
```

Simple 1 5 Julia 1.10.3 | Idle

Mode: Command Ln 1, Col 1 2024-09-10_FullSchemePropagationWithCallback.ipynb 0

Semi-automatic differentiation

$$\nabla J_T = \frac{\partial J_T}{\partial \epsilon_{nl}}$$

Automatic Differentiation:
evaluate J_T inside AD framework

Semi-automatic differentiation

$$\nabla J_T = \frac{\partial J_T(\{\Psi_k(T)\})}{\partial \epsilon_{nl}}$$

Semi-AD

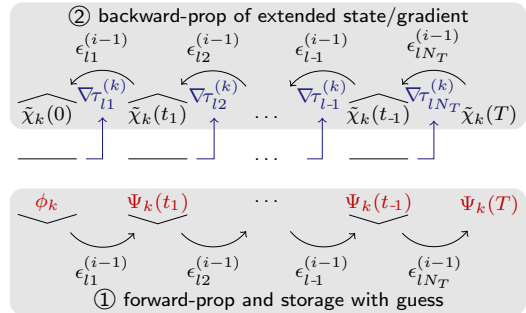
Use a chain rule to split the gradient into

- a numerically expensive but analytic part
- a non-analytic but computationally cheap part

— Goerz, Carrasco, Malinovsky. Quantum 6, 871 (2022)

Semi-automatic differentiation

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



— Goerz, Carrasco, Malinovsky. Quantum 6, 871 (2022)

NV Center Optimized Pulses



2024-09-17_SpectrumOCT_PhaseControl_TanhParam.ipynb

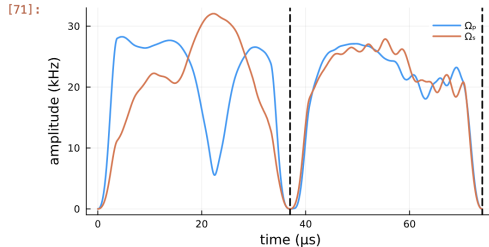
File Edit View Run Kernel Tabs Settings Help

Code

Julia 1.10.3

```
[71]: plot(tlist ./ μs, abs.(Ωp_opt) ./ kHz; label = "Ωp")
      plot!(tlist ./ μs, abs.(Ωs_opt) ./ kHz; label = "Ωs")
      vline!([T, 2T] ./ μs; color = "black", ls = :dash, label = "")
      plot!(); xlabel = "time (μs)", ylabel = "amplitude (kHz)", legend=:topright)
```

Last executed at 2024-10-28 09:29:51 in 93ms



```
[70]: plot(tlist ./ μs, angle.(Ωp_opt) ./ π; label = "arg(Ωp)")
      plot!(tlist ./ μs, angle.(Ωs_opt) ./ π; label = "arg(Ωs)")
      vline!([T, 2T] ./ μs; color = "black", ls = :dash, label = "")
      plot!(); xlabel = "time (μs)", ylabel = "phase (π)", legend=:topright)
```

Last executed at 2024-10-28 09:29:03 in 80ms

Simple 1 5 Julia 1.10.3 | Idle

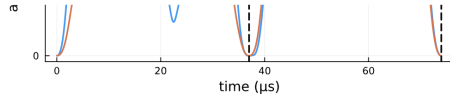
Mode: Edit Ln 2, Col 35 2024-09-17_SpectrumOCT_PhaseControl_TanhParam.ipynb 0

NV Center Optimized Pulses



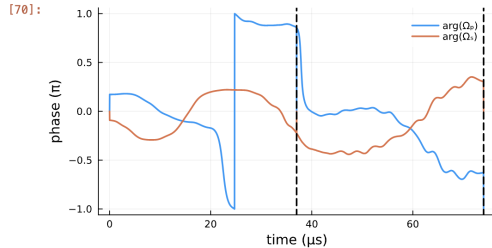
2024-09-17_SpectrumOCT_PhaseControl_TanhParam.ipynb

File Edit View Run Kernel Tabs Settings Help

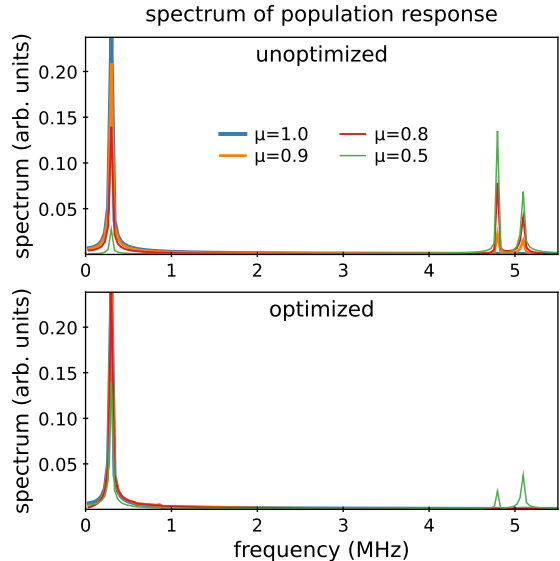
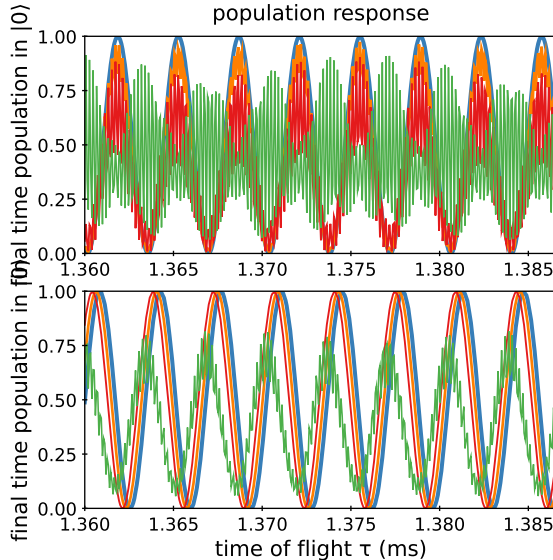
Code ⌵ gitJulia 1.10.3 ⌵

```
[70]: plot(tlist ./ μs, angle.(Ωp_opt) ./ π; label = "arg(Ωp)")
      plot!(tlist ./ μs, angle.(Ωs_opt) ./ π; label = "arg(Ωs)")
      vline!([T, 2T] ./ μs; color = "black", ls = :dash, label = "")
      plot!(); xlabel = "time (μs)", ylabel = "phase (π)", legend=:topright)
```

Last executed at 2024-10-28 09:29:03 in 80ms

Simple ⬅ 1 ⬅ 5 ⬅ Julia 1.10.3 | IdleMode: Command ✓ Ln 4, Col 26 2024-09-17_SpectrumOCT_PhaseControl_TanhParam.ipynb 0 🔔

Optimized Signal Spectrum



Conclusion

- Quantum Interferometry Implementations
 - Atomic Fountain Interferometer: robust momentum space transfer
 - Rotating Tractor Interferometer: non-adiabatic phase space transport
 - Nuclear Spin Gyroscope: “double quantum” control, spectral optimization

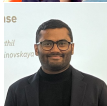
ARL



Vladimir Malinovsky



Sebastián Carrasco

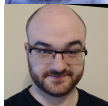


Jabir Chathanathil

Berkeley



Andrey Jarmola

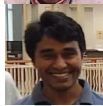


Sean Lourette

Michigan



Georg Raithel



Bineet Dash



Alisher Duspayev

Stanford



Mark Kasevich

Tim Kovachy
Remy Notermans
Chris Overstreet
Peter Asenbaum



Conclusion

- Quantum Interferometry Implementations
 - Atomic Fountain Interferometer: robust momentum space transfer
 - Rotating Tractor Interferometer: non-adiabatic phase space transport
 - Nuclear Spin Gyroscope: “double quantum” control, spectral optimization
- QuantumControl.jl Framework
 - Define control problems in terms of “trajectories”
 - Define dynamics in terms of “generators” and stateful “propagators”
 - Separate “control amplitudes” from actual “controls”
 - Efficient project-specific data structures through multiple dispatch

Thank You!

Gradients of parametrized pulses

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

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

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

<https://github.com/JuliaQuantumControl/QuantumGradientGenerators.jl>