Simulating quantum systems with Qiskit Dynamics

Daniel Puzzuoli
IBM Quantum

Dynamics contributors:
Daniel Egger, Ian Hincks, Haggai Landa, Avery Parr, Daniel Puzzuoli, Benjamin Rosand, R. K. Rupesh, Matthew Treinish, Christopher J. Wood
Outline

• Intro: Simulate what, why?
• Simulation package landscape
  • Helpful features
  • Open-source packages
• Qiskit Dynamics
  • General feature description
  • Common code examples
  • Performance comparison
  • Advanced feature: numerical perturbation theory
Simulate *what*?

- Quantum devices are quantum systems
- Schrödinger equation
  \[
  \dot{U}(t) = -iH(t)U(t)
  \]
- Lindblad equation
  \[
  \dot{\rho}(t) = -i[H(t), \rho(t)] + \sum_j L_j \rho(t)L_j^\dagger - \frac{1}{2}\{L_j^\dagger L_j, \rho(t)\}
  \]
- Others (e.g. Bloch-Redfield)
Simulation workflows

Building device models

Control design

Optimize chip design
“virtual prototyping”
Faster is better

- Faster simulation accelerates research
  - Models simulated 1000s of times
- Challenge: curse of dimensionality
  - Hilbert space dimension grows exponentially in number of subsystems
- Challenge: the other curse of dimensionality
  - Even small systems can have a lot of model parameters
  - Exploring parameter space is expensive
- Faster simulation enables more complex workflows
  - Enables asking more complicated questions
Components of a simulation package

**Workflows**
Control optimization, model fitting, ...

**Model building**
Common operators/systems, time-dependent signals, noise

**Analysis tools**
Expectation value tracking, fidelity

**Numerical Tools**
Solvers, array representation, hardware utilization

**Critical technical capabilities:**
- Speed
- Flexible/general core
- For optimization:
  - Compilation
  - Automatic differentiation
Open-source packages (non-exhaustive)

Python packages
- QuTiP (Quantum Toolbox in Python)
- $C^3$ - An integrated tool-set for control, calibration and characterization

Julia
- Quandary - Optimal control for open and closed quantum systems

C++
- qiskit-community / lindbladmpo

Matlab
- Spinach: A Fast and General Spin Dynamics Simulation Library
Qiskit Dynamics

- New Qiskit package (Python) for Hamiltonian and Lindblad simulation
- Central applications of interest
  - Optimization applications (control, model fitting)
  - Virtual prototyping
- General feature goals
  - Configurability, configurability, configurability (every problem is different)
  - Compilable and automatically differentiable
  - Integration with Qiskit ecosystem
Numerical Tools

**Solvers**
- ODE solvers
- “Geometric” solvers

**Array types**
- Dense/sparse
- Numpy/scipy/JAX

**JAX integration**
- JIT compile
- Autodiff
- GPU
Qiskit Dynamics status
Current version 0.3.0

Numerical Tools

- **Solvers**
  - ODE solvers
  - “Geometric” solvers

- **Array types**
  - Dense/sparse
  - Numpy/scipy/JAX

- **JAX integration**
  - JIT compile
  - Autodiff
  - GPU

- **Perturbation module**
  - Time-dependent perturbation theory
  - Perturbative solvers
Qiskit Dynamics status
Current version 0.3.0

Model building

General decompositions

\[ H(t) = H_0 + \sum_j s_j(t)H_j \]
\[ s_j(t) = \text{Re}[f_j(t)e^{i\omega_j t}] \]

Automated Model transformations

Rotating Frames

\[ H(t) \mapsto e^{i\mathcal{H}t}(H(t) - H_F)e^{-i\mathcal{H}t} \]

RWA: \( H(t) \approx H_{RWA}(t) \)

Numerical Tools

Solvers

"Geometric" solvers

Array types

Dense/sparse

Numpy/scipy/JAX

JAX integration

JIT compile

Autodiff

GPU

Perturbation module

Time-dependent perturbation theory

Perturbative solvers
Qiskit Dynamics status
Current version 0.3.0

Model building

General decompositions
\[ H(t) = H_0 + \sum_j s_j(t)H_j \]
\[ s_j(t) = Re[f_j(t)e^{i\omega_jt}] \]

Automated Model transformations
Rotating Frames
\[ H(t) \rightarrow e^{iH_FT}(H(t) - H_F)e^{-iH_FT} \]
RWA: \( H(t) \approx H_{RWA}(t) \)

Numerical Tools

Solvers
ODE solvers
"Geometric" solvers

Array types
Dense/sparse
Numpy/scipy/JAX

JAX integration
JIT compile
Autodiff
GPU

Qiskit Integration
Operator building
Analysis tools
Simulate Qiskit Pulse

Perturbation module
Time-dependent perturbation theory
Perturbative solvers

IBM Quantum / © 2022 IBM Corporation
Qiskit Dynamics status
Current version 0.3.0

Model building

- **General decompositions**
  \[ H(t) = H_0 + \sum_j s_j(t)H_j \]
  \[ s_j(t) = Re[f_j(t)e^{i\omega_j t}] \]

- **Automated Model transformations**
  Rotating Frames
  \[ H(t) \rightarrow e^{iH_F t}(H(t) - H_F)e^{-iH_F t} \]
  RWA: \[ H(t) \approx H_{RWA}(t) \]

Numerical Tools

- **Solvers**
  - ODE solvers
  - "Geometric" solvers

- **Array types**
  - Dense/sparse
  - Numpy/scipy/JAX

- **JAX integration**
  - JIT compile
  - Autodiff
  - GPU

IBM Quantum / © 2022 IBM Corporation

Major feature for 0.4.0
Pulse simulator
“backend”

Qiskit Integration
Operator building
Analysis tools
Simulate Qiskit Pulse

Perturbation module
Time-dependent perturbation theory
Perturbative solvers
Qiskit Dynamics status
Current version 0.3.0

Numerical Tools

**Solvers**
- ODE solvers
- “Geometric” solvers

**Array types**
- Dense/sparse
- Numpy/scipy/JAX

**JAX integration**
- JIT compile
- Autodiff
- GPU

**Perturbation module**
- Time-dependent perturbation theory
- Perturbative solvers

---

Model building

**General decompositions**

\[ H(t) = H_0 + \sum_j s_j(t)H_j \]

\[ s_j(t) = \text{Re}[f_j(t)e^{i\omega_j t}] \]

**Automated Model transformations**

- Rotating Frames
  \[ H(t) \rightarrow e^{iH_FT}(H(t) - H_F)e^{-iH_FT} \]
  \[ \text{RWA: } H(t) \approx H_{RWA}(t) \]

---

Workflows

To come

<table>
<thead>
<tr>
<th>Workflows</th>
<th>Major feature for 0.4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model building</td>
<td>Pulse simulator</td>
</tr>
<tr>
<td></td>
<td>“backend”</td>
</tr>
<tr>
<td>Workflows</td>
<td>Qiskit Integration</td>
</tr>
<tr>
<td></td>
<td>Operator building</td>
</tr>
<tr>
<td></td>
<td>Analysis tools</td>
</tr>
<tr>
<td></td>
<td>Simulate Qiskit Pulse</td>
</tr>
<tr>
<td>Numerical Tools</td>
<td>Perturbation module</td>
</tr>
<tr>
<td></td>
<td>Time-dependent</td>
</tr>
<tr>
<td></td>
<td>perturbation theory</td>
</tr>
<tr>
<td></td>
<td>Perturbative solvers</td>
</tr>
</tbody>
</table>
Simulate a Hamiltonian $H(t) = H_0 + s_1(t)H_1$

```python
Array.set_default_backend('jax')

solver = Solver(
    static_hamiltonian=H0,
    hamiltonian_operators=[H1],
    evaluation_mode='dense',
    rotating_frame=H0
)

s1 = Signal(envelope=f, carrier_freq=fdr)

result = solver.solve(
    signals=[s1],
    t_span=[times[0], times[-1]],
    y0=y0,
    t_eval=times,
    method='jax_odeint',
    atol=1e-10,
    rtol=1e-10
)
```
Getting started with Qiskit-Dynamics

• Simulate a Hamiltonian \( H(t) = H_0 + s_1(t)H_1 \)

```python
Array.set_default_backend('jax')

solver = Solver(
    static_hamiltonian=H0,
    hamiltonian_operators=[H1],
    evaluation_mode='dense',
    rotating_frame=H0
)

s1 = Signal(envelope=f, carrier_freq=fdr)

result = solver.solve(
    signals=[s1],
    t_span=[times[0], times[-1]],
    y0=y0,
    t_eval=times,
    method='jax_odeint',
    atol=1e-10,
    rtol=1e-10
)
```

Tell Dynamics to use JAX (numpy/scipy default)
Getting started with Qiskit-Dynamics

- Simulate a Hamiltonian $H(t) = H_0 + s_1(t)H_1$

```python
Array.set_default_backend('jax')

solver = Solver(
    static_hamiltonian=H0,
    hamiltonian_operators=[H1],
    evaluation_mode='dense',
    rotating_frame=H0
)

s1 = Signal(envelope=f, carrier_freq=fdr)

result = solver.solve(
    signals=[s1],
    t_span=[times[0], times[-1]],
y0=y0,
    t_eval=times,
    method='jax_odeint',
atol=1e-10,
    rtol=1e-10
)
```

Tell Dynamics to use JAX (numpy/scipy default)
Dense or sparse
Solve in arbitrary rotating frame
Getting started with Qiskit-Dynamics

- Simulate a Hamiltonian $H(t) = H_0 + s_1(t)H_1$

```python
Array.set_default_backend('jax')
tell Dynamics to use JAX (numpy/scipy default)
solver = Solver(
    static_hamiltonian=H0,
    hamiltonian_operators=[H1],
    evaluation_mode='dense',
    rotating_frame=H0
)

s1 = Signal(envelope=f, carrier_freq=fr)

result = solver.solve(
    signals=[s1],
    t_span=[times[0], times[-1]],
y0=y0,
    t_eval=times,
    method='jax_odeint',
atol=1e-10,
    rtol=1e-10
)

Tense or sparse
Solve in arbitrary rotating frame
Mixed signals with arbitrary envelopes
Getting started with Qiskit-Dynamics

- Simulate a Hamiltonian $H(t) = H_0 + s_1(t)H_1$

```python
Array.set_default_backend('jax')  # Tell Dynamics to use JAX (numpy/scipy default)
solver = Solver(
    static_hamiltonian=H0,
    hamiltonian_operators=[H1],
    evaluation_mode='dense',
    rotating_frame=H0
)
s1 = Signal(envelope=f, carrier_freq=fdr)
result = solver.solve(
    signals=[s1],
    t_span=[times[0], times[-1]],
    y0=y0,
    t_eval=times,
    method='jax_odeint',
    atol=1e-10,
    rtol=1e-10
)
```

Dense or sparse

Solve in arbitrary rotating frame

Mixed signals with arbitrary envelopes

Solver choice and options
Qiskit Pulse simulation

- Can configure Solver to solve Qiskit Pulse schedules

```python
# construct the solver
hamiltonian_solver = Solver(
    static_hamiltonian=drift,
    hamiltonian_operators=operators,
    rotating_frame=drift,
    rwa_cutoff_freq=2 * 5.0,
    hamiltonian_channels=['d0'],
    channel_carrier_freqs={'d0': w},
    dt=dt
)

with pulse.build(name="sx-sy schedule") as xp:
    pulse.play(pulse.Drag(duration, amp / 1.75, sig / dt, beta), pulse.DriveChannel(0))
    pulse.shift_phase(np.pi / 2, pulse.DriveChannel(0))
    pulse.play(pulse.Drag(duration, amp / 1.75, sig / dt, beta), pulse.DriveChannel(0))

xp.draw()

%time sol = hamiltonian_solver.solve(t_span=[0., 2*T], y0=y0, signals=xp, atol=1e-8,
rtol=1e-8)
```

Add pulse configuration information to model structure

Pass schedule to solve
QuTiP v.s. Dynamics speed comparisons

- Using Dynamics to tune gate parameters in a 3 transmon model
- Dimension 160
- 1000 sims on GPU w/vmap $\approx 147$ s
  - Equiv to $\approx 25$ sims in parallel at single sim speed
Advanced feature: Perturbation theory

- Numerical perturbation theory module added in 0.3.0
  - Algorithms for perturbative analysis and simulation of quantum dynamics (arxiv.org/abs/2210.11595)
    D. Puzzuoli, S. F. Lin, M. Malekakhlagh, E. Pritchett, B. Rosand, C. J. Wood

- Perturbative solvers, based on Dyson series and Magnus expansion
  - Simulating 100 different pulse parameter values for 2 qubit gate on GPU for system with dimension 25

Variation of solvers in:
Closing

• Package overview recap
• Current version 0.3.0: Core numerical foundation, automatic model transformations, JIT, autodiff
• Future version 0.4.0: Full Qiskit Pulse integration, simulator backend
• Beyond: Tools for building optimization workflows
• Please submit issues, ask for help, and/or contribute!
• Github: [github.com/Qiskit/qiskit-dynamics](https://github.com/Qiskit/qiskit-dynamics)
• Documentation: [qiskit.org/documentation/dynamics/](https://qiskit.org/documentation/dynamics/)
• Slack: Qiskit workspace, [#qiskit-dynamics](https://qiskit.org/discord)
• Thank you!
Workshop closing
Thanks to all our participants and attendees!

Jens Koch
Katarina Cicak
Angela Kou
John Teufel
Archana Kamal
Amir Safavi-Naeini
Holger Haas
Michael Hush
Ziwen Huang

Thanks to all my co-organizers:
Yehan Liu, Daniel Puzzuoli, and Patrick O’Brien