QuTiP: An open-source Python framework for the dynamics of open quantum systems

J. R. Johansson[a,1,*], P. D. Nation[a,b,1,*], Franco Nori[a,b]

[a]*Advanced Science Institute, RIKEN, Wako-shi, Saitama 351-0198, Japan*
[b]*Department of Physics, University of Michigan, Ann Arbor, Michigan 48109-1040, USA*

Abstract

We present an object-oriented open-source framework for solving the dynamics of open quantum systems written in Python. Arbitrary Hamiltonians, including time-dependent systems, may be built up from operators and states defined by a quantum object class, and then passed on to a choice of master equation or Monte-Carlo solvers. We give an overview of the basic structure for the framework before detailing the numerical simulation of open system dynamics. Several examples are given to illustrate the build up to a complete calculation. Finally, we measure the performance of our library against that of current implementations. The framework described here is particularly well-suited to the fields of quantum optics, superconducting circuit devices, nanomechanics, and trapped ions, while also being ideal for use in classroom instruction.

Keywords: Open quantum systems, Lindblad master equation, Quantum Monte-Carlo, Python
PACS: 03.65.Yz, 07.05.Tp, 01.50.hv

PROGRAM SUMMARY

*Corresponding authors
Email addresses: robert@riken.jp (J. R. Johansson), pnation@riken.jp (P. D. Nation)
[1]These authors contributed equally to this work.

1. Introduction

Every quantum system encountered in the real world is an open quantum system [1]. For although much care is taken experimentally to eliminate the unwanted influence of external interactions, there remains, if ever so slight, a coupling between the system of interest and the external world. In addition, any measurement performed on the system necessarily involves coupling to the measuring device, therefore introducing an additional source of external influence. Consequently, developing the necessary tools, both theoretical and numerical, to account for the interactions between a system and its environment is an essential step in understanding the dynamics of quantum systems.

By definition, an open quantum system is coupled to an environment, also called a reservoir or bath, where the complexity of the environmental dynamics renders the combined evolution of system plus reservoir intractable. However, for a system weakly coupled to its surroundings, there is a clear distinction between the system and its environment, allowing for the dynamics of the environment to be traced over, resulting in a reduced density matrix describing the system alone. The most general dynamical equation governing this reduced system density matrix is given by the Lindblad master equation [2] describing the evolution of an ensemble average of a large (formally infinite) number of identical system realizations. Although the density operator formalism sufficed for the first half-century of quantum mechanics, the advent of single-ion traps in the 1980's [3] motivated the study of the quantum

trajectories, or Monte-Carlo, description for the evolution of a *single* realization of a dissipative quantum system [4].

In general, for all but the most basic of Hamiltonians, an analytical description of the system dynamics is not possible, and one must resort to numerical simulations of the equations of motion. In absence of a quantum computer [5], these simulations must be carried out using classical computing techniques, where the exponentially increasing dimensionality of the underlying Hilbert space severely limits the size of system that can be efficiently simulated [6, 7]. However, in many fields such as quantum optics [8, 9], trapped ions [3, 10], superconducting circuit devices [11, 12, 13], and most recently nanomechanical systems [14, 15, 16, 17], it is possible to design systems using a small number of effective oscillator and spin components, excited by a small number of quanta, that are amenable to classical simulation in a truncated Hilbert space.

Of the freely available quantum evolution software packages [18, 19, 20], the Quantum Optics Toolbox (qotoolbox) [19] has by far been the most successful. Although originally geared toward quantum optics, the qotoolbox has gained popularity in a variety of disciplines, driven in part by its rapid modeling capabilities and easy to read code syntax. Yet, at the same time, the qotoolbox has not been updated in nearly a decade, leaving researchers to rely on an outdated numerical platform. Moreover, while the code underlying the qotoolbox is open-sourced, it does rely on the proprietary Matlab [21] computing environment making it an impractical solution for many research groups, as well as for use as an educational tool inside the classroom.

In this paper, we describe a fully open-source implementation of a framework designed for simulating open quantum dynamics written in the Python programming language [22] called the *Quantum Toolbox in Python* or QuTiP [23]. This framework distinguishes itself from the other available software solutions by providing the following advantages:

- Based entirely on open-source software.

- Easy to read, rapid code development using the Python programming language.

- Support for arbitrary, time dependent Hamiltonians.

- Makes use of the multiple processing cores found in modern computers.

- Community based infrastructure, allowing for user contributions to the code base.

Although Python is an interpreted programming language, it is well suited for scientific computations as a result of its large collection of high-performance low-level numerical libraries [24], mathematical functions [25], and data visualization capabilities [26], that largely are implemented in efficient compiled code. In particular, QuTiP relies heavily on the sparse matrix and dense array functionality provided by the *SciPy* [25] and *NumPy* [24] packages, respectively. Since the bulk of a typical calculation is spent in these libraries, a QuTiP simulation can achieve nearly the same performance as compiled code. The advantage of using the Python programming language over a compiled programming language is a greatly simplified development process, and more transparent, less bug-prone code. For data visualization QuTiP uses the matplotlib package [26], which is capable of producing publication-quality 2D and 3D figures in a wide range of styles and formats.

Given the success of the qotoolbox, the development of QuTiP has in part been directed toward providing a replacement for this excellent, yet aging software. In the spirit of open-source development, we have strived to use the best parts of the qotoolbox in QuTiP, while improving, replacing, or complementing the parts that were in need of modernization. The result is a framework for simulating quantum system dynamics that is in many ways more efficient and better suited for modern computers, as well as better positioned for further development and adoption to new computer architecture advances. Given the size of the QuTiP framework, we do not hope to cover all of its functionality here. Instead, we will focus on the key data structures, and numerical routines underlying the majority of calculations. In addition, we will highlight a variety of example calculations that we hope will give the reader a flavor of the capabilities of QuTiP, and highlight what is possible using this framework. A complete overview of QuTiP is given on its website [23].

This paper is organized as follows. In Sec. 2 we introduce the main QuTiP class, representing a quantum operator or state vector, and its associated data structures and methods. In Sec. 3 we give a brief overview of the density matrix formalism before discussing the master equation and Monte-Carlo methods used in QuTiP. Section 4 presents a selection of examples meant to illustrate how calculations are performed using the QuTiP framework. Section 5 compares the performance of the QuTiP master equation and Monte-Carlo solvers to those in the qotoolbox. Finally, Sec. 6 briefly concludes, while a list of user accessible functions built into QuTiP, as well as example codes, are relegated to the appendix.

2. The QuTiP framework

QuTiP provides an object-oriented framework for representing generic quantum systems, and for performing calculations and simulations on such systems. In order to simulate a quantum system, one must first construct an object that encapsulates the properties of an arbitrary state vector or operator. A unified representation of quantum operators and state vectors is implemented in QuTiP by means of the quantum object class (Qobj), that uses a sparse matrix representation of a quantum object in a
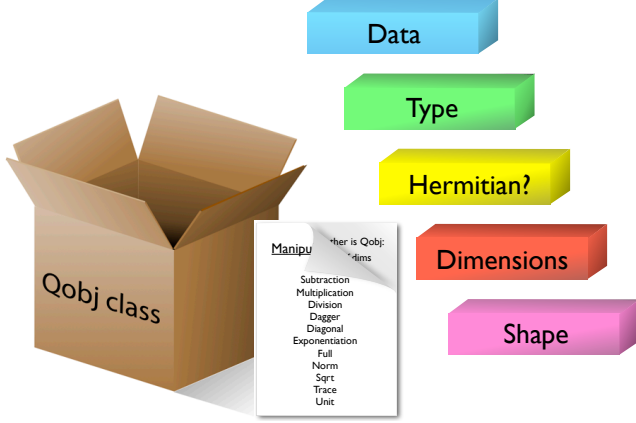
Figure 1: (Color) A quantum object, operator or state vector, is represented by a quantum object class (Qobj) instance. The Qobj class may be thought of as a container, holding the data structures required to fully characterize a generic quantum object, as well as a list of instructions on how to manipulate these items. The primary data structures are the data, in sparse matrix form, that represents a quantum object in a given Hilbert space, the type of the object (ket, bra, operator, super-operator), whether it is Hermitian or not, and the objects dimension and shape. Here, the dimension describes the structure of the Hilbert space, i.e., whether it is a composite system and how it is composed. The Qobj class also defines a variety of methods that implement common functions operating on quantum objects. See Table 1 for a list of the Qobj class methods.

finite dimensional Hilbert space. The Qobj class internally maintains a record of the principal attributes of the quantum object it represents. These include, the objects type (i.e ket, bra, operator, or super-operator), whether the underlying object is Hermitian, the dimensionality of a composite object formed via the tensor-product, and the size of the sparse data matrix. A schematic illustration of the key components underlying the Qobj class is shown in Fig. 1.

In addition to serving as a book-keeper for the properties of a quantum object, the Qobj class is also a computational object, implementing the usual binary arithmetic operations, and a variety of class methods for performing common object manipulations as presented in Table 1. Therefore, with just a few lines of QuTiP code, it is

Method	Description
`dag()`	Adjoint of the quantum object.
`diag()`	Diagonal elements of object.
`eigenstates()`	Eigenstates and eigenvectors.
`expm()`	Exponentiated quantum object.
`full()`	Dense array representation.
`norm()`	L2 norm (states), trace norm (oper).
`sqrtm()`	Matrix square root.
`tr()`	Trace of quantum object.
`unit()`	Normalizes the quantum object.

Table 1: List of methods built into the Qobj class.

straightforward to construct Hamiltonians from arbitrary combinations of operators, and to construct density matrices and state vectors that represent complicated superpositions of basis states. To further simplify this important step, QuTiP provides a library of commonly occurring operators and states which are given in Appendix A.

For example, to create an instance of the Qobj class that represents the ubiquitous two-level Hamiltonian ($\hbar = 1$)

$$ H = \frac{1}{2}\epsilon\sigma_z + \frac{1}{2}\Delta\sigma_x, \tag{1} $$

with energy splitting ϵ and transition energy Δ, one can use the following QuTiP code:

```
H = 0.5 * epsilon * sigmaz() + 0.5 * delta * sigmax()
```

where `epsilon` and `delta` represent user defined constants. The result is a single Qobj instance H that represents the Hamiltonian operator Eq. (1).

Composite quantum systems are nearly as easy to create. Consider the Jaynes-Cumming Hamiltonian

$$ H = \omega_0 a^\dagger a + \frac{1}{2}\epsilon\sigma_z + g(a^\dagger\sigma_+ + a\sigma_-), \tag{2} $$

with cavity frequency ω_0 and coupling strength g, describing a cavity field coupled to a two-level atom (qubit). A Qobj instance representing this composite Hamiltonian can be created with the following code:

```
a  = tensor(destroy(N), qeye(2))
sm = tensor(qeye(N), destroy(2))
sz = tensor(qeye(N), sigmaz())
H  = omega0 * a.dag() * a + 0.5 * epsilon * sz
     + g * (a.dag() * sm + a * sm.dag())
```

where the `tensor` function is used to construct composite operators for the combined Hilbert space of the cavity (truncated to the N lowest Fock states) and the atom.

Since the Qobj class provides a unified representation for operators and states, we can use exactly the same technique to generate quantum states (either as state vectors or density matrices). A possible initial state for the system described by Eq. (2), is generated with the QuTiP code:

```
psi0 = tensor(fock(N,0),(fock(2,0)+fock(2,1)).unit())
```

creating the Qobj representation of a cavity in its ground state, coupled to a qubit in a balanced superposition of its ground and excited state $|\Psi(0)\rangle = (|0\rangle_c|0\rangle_q + |0\rangle_c|1\rangle_q)/\sqrt{2}$. Here, the subscripts c and q denote the cavity and qubit states, respectively. The normalization factor ($\sqrt{2}$) is applied automatically using the `unit()` method.

In the previous example, we have used builtin QuTiP library functions to generate Qobj instances of commonly occurring operators and states, and the associated arithmetic operations in the Qobj class[2] to combine these quantum operators into more complicated systems. The

---

[2]The binary arithmetic operators +, -, and * are defined for two Qobj objects, and +, -, * as well as / are defined between a Qobj object and a real or complex number.

close correspondence between the mathematical formulation and the programming code makes it easy and transparent to define quantum objects. This is especially important when working with quantum systems of more complex structure. Note that the `Qobj` instances in the previous examples are all self-contained descriptions of the quantum object they represent. From the `Qobj` instance alone, a number of properties pertaining to the quantum object may be calculated, and various operations be applied (see Table 1). This includes, for example, operations such as the Hermitian adjoint, normalization, and trace, as well as computation of the eigenstates and eigenenergies and operator exponentiation.

In addition, QuTiP includes several important functions operating on multiple states and/or operators (see Table A.2). We have already seen one such example in the `tensor` function used to generate tensor product states. These states may also be decomposed into their constituent parts by performing a partial trace over selected degrees of freedom using the function `ptrace`. For example, from the composite wave function `psi0` for the oscillator-qubit system, the state of the qubit can be extracted by tracing out the oscillator degrees of freedom using the QuTiP code:

```
rho0_qubit = ptrace(psi0, 1)
```

where the second argument is the index of the system that we wish to *keep*. In general, it can be a list of indices. The properties of the resulting `Qobj` instance (shown in Fig. 1) may be inspected by displaying the string representation of the object returned by its `str` method. This method is implicitly invoked when the object is printed to the standard output

```
print rho0_qubit
Quantum object: dims = [[2], [2]], shape = [2, 2],
type = oper, isHerm = True
Qobj data =
[[ 0.5  0.5]
 [ 0.5  0.5]]
```

which, in this case, shows that the `Qobj` instance `rho0_qubit` is a 2×2, Hermitian quantum operator representing a balanced coherent superposition of its two basis states. From a Qobj instance one may also calculate items such as the expectation value (`expect`) for an arbitrary operator with the QuTiP function, find the fidelity (`fidelity`) between two density matrices [27], or calculate the Wigner function (`wigner`) of a quantum state. Using these, and other functions (Table. A.2), in the exploration of open quantum dynamics will be the focus of Sec. 4.

Even though the emphasis of QuTiP is on dynamical modeling, it is possible to obtain nontrivial results directly from a quantum object. As an example, let us consider the Jaynes-Cummings model in the ultra-strong coupling regime $g \geq \omega_0, \epsilon$ where the rotating wave approximation (RWA) is no longer valid

$$H = \omega_0 a^\dagger a + \frac{1}{2}\epsilon\sigma_z + g\left(a^\dagger + a\right)\left(\sigma_+ + \sigma_-\right). \qquad (3)$$

Recently, this regime has become of interest [28, 29, 30] due to the experimental realization of the required large
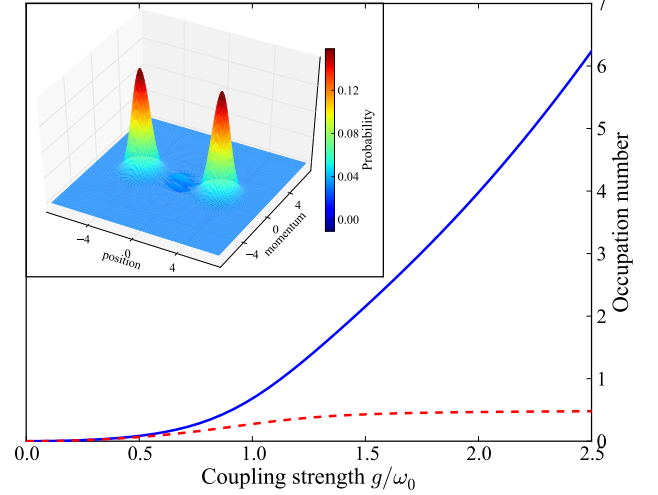

Figure 2: (Color) Expectation value for the number of excitations in the cavity (blue) and qubit (dashed-red) modes of the non-RWA Jaynes-Cummings model Eq. (3) as the coupling strength g is increased into the ultra-strong coupling regime $g/\omega_0 \gtrsim 1$. The inset figure displays the Wigner function for the cavity mode at the largest coupling strength, $g = 2.5\omega_0$, where ω_0 is the bare cavity frequency. At this coupling value, the state of the system is well-approximated by Eq. (4).

coupling strengths in superconducting circuit devices [31]. When the coupling strength g is a significant fraction of the cavity and qubit frequencies, the ground state of the cavity mode, after tracing out the qubit, is no longer the vacuum state. Instead, the anti-resonant terms proportional to $a^\dagger\sigma_+$ and $a\sigma_-$ give rise to an anomalous ground state which, in the large coupling limit $g/\omega_0 \gg 1$, may be approximated as [32, 29]

$$|\psi_g\rangle \simeq \frac{1}{\sqrt{2}}\left[|\alpha\rangle_c|+\rangle_q - |-\alpha\rangle_c|-\rangle_q\right], \qquad (4)$$

where the cavity mode is in a Schrödinger cat-state with $|\alpha| \simeq g$. This ground state can be evaluated by finding the eigenstates and eigenvalues of the Hamiltonian, and can therefore be extracted directly from the `Qobj` representation of Eq. (3). In Fig. (2) we plot the cavity and qubit occupation numbers for the groundstate of Eq. (3) as a function of the coupling strength. Here, the cavity is on resonance with the qubit transition frequency, $\omega_0 = \epsilon = 2\pi$. In addition, Fig. 2 shows the Wigner function for the cavity mode at the largest coupling strength $g = 2.5\omega_0$, which is well approximated by Eq. (4). The 20 lines of QuTiP code used in calculating Fig. 2 are given in Appendix B.1.

3. Evolution of open quantum systems

The main focus of QuTiP is the time-evolution of open quantum systems. Before we describe how this problem is approached in QuTiP, we give a brief review of the theory of quantum evolution, and the available methods for numerically integrating the equations of motion.

The dynamics of a closed (pure) quantum system is governed by the Schrödinger equation

$$i\hbar\frac{\partial}{\partial t}\Psi = \hat{H}\Psi, \qquad (5)$$

where Ψ is the wave function, $\hat{H}$ the Hamiltonian, and $\hbar$ is Planck's constant. In general, the Schrödinger equation is a partial differential equation (PDE) where both Ψ and $\hat{H}$ are functions of space and time. For computational purposes it is useful to expand the PDE in a set of basis functions that span the Hilbert space of the Hamiltonian, and to write the equation in matrix and vector form

$$i\hbar\frac{d}{dt}\left|\psi\right\rangle = H\left|\psi\right\rangle, \qquad (6)$$

where $\left|\psi\right\rangle$ is the state vector and H is the matrix representation of the Hamiltonian. This matrix equation can, in principle, be solved by diagonalizing the Hamiltonian matrix H. In practice, however, it is difficult to perform this diagonalization unless the size of the Hilbert space (dimension of the matrix H) is small. Analytically, it is a formidable task to calculate the dynamics for systems with more than two states. If, in addition, we consider dissipation due to the inevitable interaction with a surrounding environment, the computational complexity grows even larger, and we have to resort to numerical calculations in all realistic situations. This illustrates the importance of numerical calculations in describing the dynamics of open quantum systems, and the need for efficient and accessible tools for this task.

While the evolution of the state vector in a closed quantum system is deterministic, open quantum systems are stochastic in nature. The effect of an environment on the system of interest is to induce stochastic transitions between energy levels, and to introduce uncertainty in the phase difference between states of the system. The state of an open quantum system is therefore described in terms of ensemble averaged states using the density matrix formalism. A density matrix ρ describes a probability distribution of quantum states $\left|\psi_n\right\rangle$, in a matrix representation $\rho = \sum_n p_n \left|\psi_n\right\rangle\left\langle\psi_n\right|$, where p_n is the classical probability that the system is in the quantum state $\left|\psi_n\right\rangle$. The time evolution of a density matrix ρ is the topic of the remaining portions of this section.

3.1. Master equation

The standard approach for deriving the equations of motion for a system interacting with its environment is to expand the scope of the system to include the environment. The combined quantum system is then closed, and its evolution is governed by the von Neumann equation

$$\dot{\rho}_{\text{tot}}(t) = -\frac{i}{\hbar}[H_{\text{tot}}, \rho_{\text{tot}}(t)], \qquad (7)$$

the equivalent of the Schrödinger equation (5) in the density matrix formalism. Here, the total Hamiltonian

$$H_{\text{tot}} = H_{\text{sys}} + H_{\text{env}} + H_{\text{int}}, \qquad (8)$$

includes the original system Hamiltonian H_{sys}, the Hamiltonian for the environment H_{env}, and a term representing the interaction between the system and its environment H_{int}. Since we are only interested in the dynamics of the system, we can at this point perform a partial trace over the environmental degrees of freedom in Eq. (7), and thereby obtain a master equation for the motion of the original system density matrix. The most general trace-preserving and completely positive form of this evolution is the Lindblad master equation for the reduced density matrix $\rho = \text{Tr}_{\text{env}}[\rho_{\text{tot}}]$

$$\dot{\rho}(t) = -\frac{i}{\hbar}[H(t), \rho(t)]$$
$$+ \sum_n \frac{1}{2}\left[2C_n\rho(t)C_n^\dagger - \rho(t)C_n^\dagger C_n - C_n^\dagger C_n\rho(t)\right], \quad (9)$$

where the $C_n = \sqrt{\gamma_n}A_n$ are collapse operators, and A_n are the operators through which the environment couples to the system in H_{int}, and γ_n are the corresponding rates. The derivation of Eq. (9) may be found in several sources [2, 33, 34], and will not be reproduced here. Instead, we emphasize the approximations that are required to arrive at the master equation in the form of Eq. (9), and hence perform a calculation in QuTiP:

Separability: At $t = 0$ there are no correlations between the system and its environment such that the total density matrix can be written as a tensor product $\rho_{\text{tot}}^I(0) = \rho^I(0) \otimes \rho_{\text{env}}^I(0)$.

Born approximation: Requires: (1) that the state of the environment does not significantly change as a result of the interaction with the system; (2) The system and the environment remain separable throughout the evolution. These assumptions are justified if the interaction is weak, and if the environment is much larger than the system. In summary, $\rho_{\text{tot}}(t) \approx \rho(t) \otimes \rho_{\text{env}}$.

Markov approximation: The time-scale of decay for the environment τ_{env} is much shorter than the smallest time-scale of the system dynamics $\tau_{\text{sys}} \gg \tau_{\text{env}}$. This approximation is often deemed a "short-memory environment" as it requires that environmental correlation functions decay on a time-scale fast compared to those of the system.

Secular approximation: Stipulates that elements in the master equation corresponding to transition frequencies satisfy $|\omega_{ab} - \omega_{cd}| \ll 1/\tau_{\text{sys}}$, i.e., all fast rotating terms in the interaction picture can be neglected. It also ignores terms that lead to a small renormalization of the system energy levels. This approximation is not strictly necessary for all master-equation formalisms (e.g., the Block-Redfield master equation), but it is required for arriving at the Lindblad form (9) which is used in QuTiP.

For systems with environments satisfying the conditions outlined above, the Lindblad master equation (9) governs the time-evolution of the system density matrix, giving an ensemble average of the system dynamics. In order to ensure that these approximations are not violated, it is important that the decay rates γ_n be smaller than the minimum energy splitting in the system Hamiltonian. Situations that demand special attention therefore include, for example, systems strongly coupled to their environment, and systems with degenerate or nearly degenerate energy levels.

In QuTiP there are two solvers that calculate the time evolution according to Eq. (9): `odesolve` numerically integrates the set of coupled ordinary differential equations (ODEs), and `essolve` which employs full diagonalization. The `odesolve` and `essolve` solvers both take the same set of input parameters (as exemplified in Sec. 4) and can easily be substituted for each other in a QuTiP program. For a quantum system with N states, the number of elements in the density matrix is N^2, and solving the master equation by numerical integration or diagonalization involves of use of superoperators of size $N^2 \times N^2$. In the sparse matrix format, not all of the N^4 elements need to be stored in the memory. However, the time required to evolve a quantum system according to the master equation still increases rapidly as a function of the system size. Consequently, the master equation solvers are practical only for relatively small systems: $N \lesssim 1000$, depending on the details of the problem. In Fig. 3 we show the scaling of the elapsed time for a typical simulation, here chosen to be the Heisenberg spin-chain

$$H = -\frac{1}{2} \sum_n^M h^n \sigma_z^n - \frac{1}{2} \sum_n^{M-1} \left[J_x^n \sigma_x^n \sigma_x^{n+1} \right. \tag{10}$$
$$\left. + J_y^n \sigma_y^n \sigma_y^{n+1} + J_z^n \sigma_z^n \sigma_z^{n+1} \right],$$

as a function of the size of the Hilbert space, for the two master equation solvers, as well as for two realizations of the Monte-Carlo solver `mcsolve` described the following section. In general, the exact time required to evolve a system depends on the details of the problem, but the scaling with system size is rather generic. The Monte-Carlo solver has superior scaling properties compared to the master-equation solvers, but due to the overhead from stochastic averaging, it is only for systems with a Hilbert space dimension around ~ 1000 that the Monte-Carlo solvers outperform the master equation.

3.2. Monte-Carlo trajectories

Where as the density matrix formalism describes the ensemble average over many identical realizations of a quantum system, the Monte-Carlo (MC), or quantum-jump approach [4] to wave function evolution, allows for simulating an individual realization of the system dynamics. Here, the environment is continuously monitored, resulting in a series of quantum jumps in the system wave function,
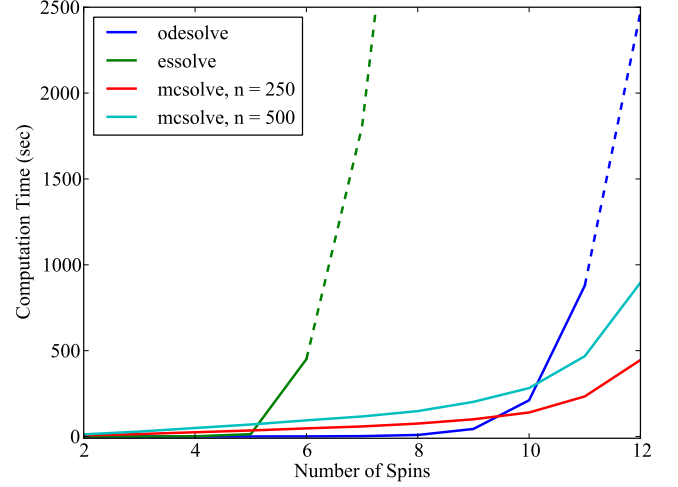

Figure 3: (Color) The time required to evolve the Heisenberg spin-chain, Eq. (10), as a function of the system size 2^M where M is the number of spins, using the master equation ODE solver `odesolve` (blue), diagonalization via `essolve` (green), and the Monte-Carlo solver `mcsolve` with 250 (red) and 500 (cyan) trajectories, respectively. The dashed lines give the estimated calculation times extrapolated from the data when the simulation could no longer fit in the computers memory (odesolve), or the calculation became intractable (essolve). Here, the spin parameters are assumed to be identical with $h = 2\pi$ and $J_x = J_y = J_z = 0.1 \times 2\pi$. Likewise, each spin has a dephasing rate given by $\gamma = 0.01$. The initial state is given by $|\psi(0)\rangle = |1\rangle_1 |0\rangle_2 \ldots |0\rangle_M$. Calculations were performed on a 2.8 Ghz quad-core computer with 24 Gb of memory.

conditioned on the increase in information gained about the state of the system via the environmental measurements [8]. In general, this evolution is governed by the Schrödinger equation (5) with a *non-Hermitian* effective Hamiltonian

$$H_{\text{eff}} = H_{\text{sys}} - \frac{i\hbar}{2} \sum_i C_n^\dagger C_n, \tag{11}$$

where again, the C_n are collapse operators, each corresponding to a separate irreversible process with rate γ_n. Here, the strictly negative non-Hermitian portion of Eq. (11) gives rise to a reduction in the norm of the wave function, that to first-order in a small time δt, is given by $\langle \psi(t+\delta t)|\psi(t+\delta t)\rangle = 1 - \delta p$ where

$$\delta p = \delta t \sum_n \left\langle \psi(t)|C_n^\dagger C_n|\psi(t)\right\rangle, \tag{12}$$

and δt is such that $\delta p \ll 1$. With a probability of remaining in the state $|\psi(t+\delta t)\rangle$ given by $1-\delta p$, the corresponding quantum jump probability is thus Eq. (12). If the environmental measurements register a quantum jump, say via the emission of a photon into the environment [35], or a change in the spin of a quantum dot [36], the wave function undergoes a jump into a state defined by projecting $|\psi(t)\rangle$ using the collapse operator C_n corresponding to the

measurement

$$|\psi(t + \delta t)\rangle = C_n |\psi(t)\rangle / \langle\psi(t)|C_n^\dagger C_n|\psi(t)\rangle^{1/2}. \quad (13)$$

If more than a single collapse operator is present in Eq (11), the probability of collapse due to the ith-operator C_i is given by

$$P_i(t) = \left\langle\psi(t)|C_i^\dagger C_i|\psi(t)\right\rangle/\delta p. \quad (14)$$

Evaluating the MC evolution to first-order in time is quite tedious. Instead, QuTiP uses the following algorithm to simulate a single realization of a quantum system [37, 38, 39]. Starting from a pure state $|\psi(0)\rangle$:

I: Choose a random number r between zero and one, representing the probability that a quantum jump occurs.

II: Integrate the Schrödinger equation (5), using the effective Hamiltonian (11) until a time τ such that the norm of the wave function satisfies $\langle\psi(\tau)\,|\psi(\tau)\rangle = r$, at which point a jump occurs.

III: The resultant jump projects the system at time τ into one of the renormalized states given by Eq. (13). The corresponding collapse operator C_n is chosen such that n is the smallest integer satisfying

$$\sum_{i=1}^{n} P_n(\tau) \geq r \quad (15)$$

where the individual P_n are given by Eq. (14). Note that the left hand side of Eq. (15) is, by definition, normalized to unity.

IV: Using the renormalized state from step III as the new initial condition at time τ, draw a new random number, and repeat the above procedure until the final simulation time is reached.

3.2.1. Example: Single-photon cavity decay

As an illustrative example, let us consider the evolution of a single-photon cavity Fock state in a non-zero thermal environment [40]. The evolution of the wave function $|\psi(0)\rangle = |1\rangle$ is governed by the effective Hamiltonian ($\hbar = 1$)

$$H_{\text{eff}} = \omega_c a^\dagger a - \frac{1 + \langle n\rangle_{\text{th}}}{2} i\kappa a^\dagger a - \frac{\langle n\rangle_{\text{th}}}{2} i\kappa a a^\dagger, \quad (16)$$

with cavity frequency ω_c, cavity decay rate κ, and where $\langle n\rangle_{\text{th}}$ is the steady state thermal occupation number. While the first term in Eq. (11) is responsible for the standard unitary evolution of the cavity mode, the second and third terms give rise to random quantum jumps to lower and higher cavity photon numbers, respectively. When a jump occurs, the wave function of the system is projected into a state corresponding to the collapse operator
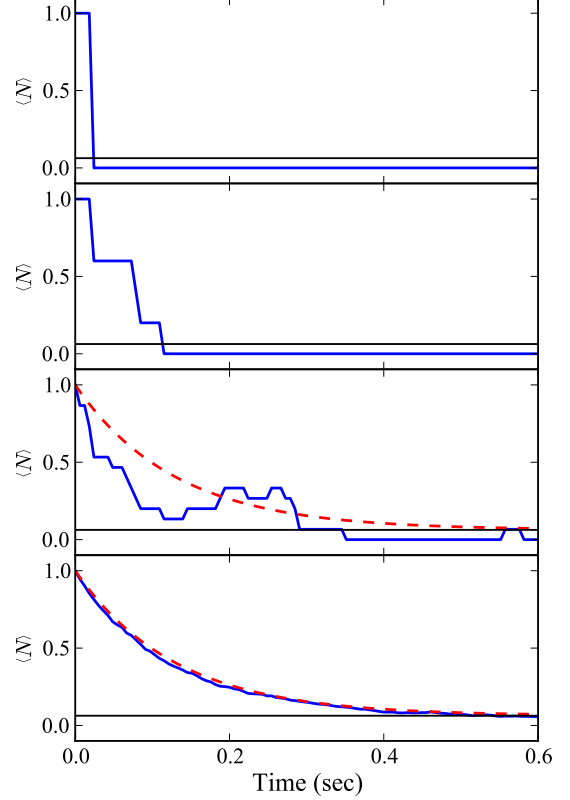

Figure 4: (Color) (a) Monte-Carlo simulation showing the number operator expectation value for a single trajectory (blue) in the decay of a single-photon Fock state from a cavity coupled to a thermal environment, as demonstrated experimentally in Ref. [40]. Here, the average photon number for the thermal environment is $n = 0.063$ (black), while the decay rate $\kappa = 1/T_c$ is given by the cavity ring-down time $T_c = 0.129$. (b-d) Averages of 5, 15, and 904 trajectories showing ensemble averaging toward the master equation solution (dashed-red).

$C_1 = \sqrt{(1 + \langle n\rangle_{\text{th}})\kappa}\,a$, yielding a decrease in the cavity occupation number, or $C_2 = \sqrt{\langle n\rangle_{\text{th}}\kappa}\,a^\dagger$, which results in an increase. Here, the relative ratio of jumps corresponding to an increase in the cavity occupation number to those for decay is determined by the magnitude of $\langle n\rangle_{\text{th}}$. A single realization of this evolution, showing a lone quantum jump, is presented in Fig. 4a.

In addition to single quantum systems, when averaged over a sufficiently large number of identical system realizations, the MC method leads to the same evolution equation as the Lindblad master equation (9) for a pure state density matrix [4, 8]. Therefore, the MC method may be used in any situation where the Lindblad master equation is valid, as discussed in Sec. 3.1. However, for large quantum systems with Hilbert space dimension $N \gg 1$, the MC method is vastly more efficient than simulating the full density matrix given that only N elements are required to simulate a wave function, as opposed to the N^2 elements necessary in the ME approach. Although multiple trajectories are required, convergence to the ME result scales as m^{-1} [34], where m is the number of trajectories simulated.
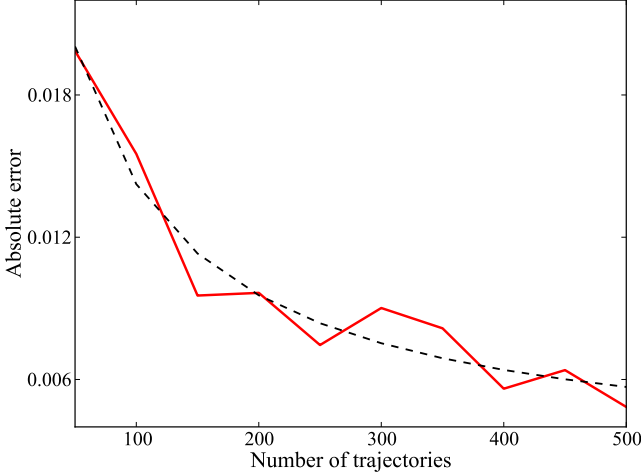
7

Figure 5: (Color) Scaling of the sum of the absolute deviation per time step, averaged over 10 simulations, for expectation values calculated with the Monte-Carlo solver and compared to the ME solution, as a function of the number of trajectories. The error measure is calculated by integrating the absolute errors over the entire evolution time, and normalized by the number of time steps. The acceptable level of error varies from case to case, but in general the error measure should be much smaller than unity. A $1/m$ fit to the data (dashed) showing the predicted convergence rate is also presented.

In typical situations, between 250 and 500 trajectories are sufficient for errors smaller than a few percent, see Fig. 5. In Fig. 4 we show the convergence of the MC simulation to the ME solution for the single-photon cavity example as the number of trajectories averaged over is increased.

4. Numerical calculations

In this section we illustrate, via a number of examples, how quantum dynamical calculations are carried out using the QuTiP framework. The typical workflow for performing a simulation with qutip is:

I: Define the parameters that characterize the system and environment (if applicable).

II: Create `Qobj` class instances representing the Hamiltonian and initial state of the system.

III: For dissipative systems, define the collapse operators as `Qobj` objects.

IV: Evolve the system with a choice of evolution algorithm and output (e.g., operator expectation values).

V: Post-process and visualize the data.

Using the quantum object described in Sec. 2, and the solvers for the time-evolution of quantum systems described in Sec. 3, we can explore a diverse set of problems. The examples presented here are selected because they illustrate different features of QuTiP with a minimum of complexity.
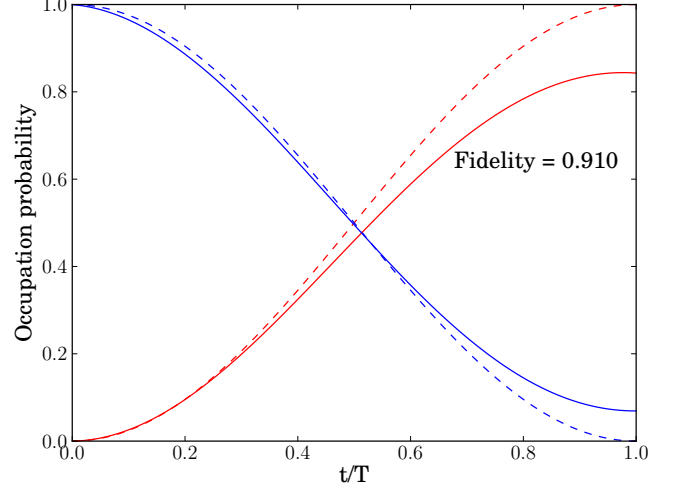

Figure 6: (Color) The time-evolution of a two-qubit system described by the Hamiltonian Eq. (17), which at time $t/T = 1$ and ideal conditions (dashed lines) transforms the qubit states in accordance with the i-SWAP gate. With qubit relaxation and dephasing (solid lines), the end-result deviates from the ideal i-SWAP gate. For these particular parameters ($g = 2\pi$, $\Gamma_1 = 0.75$, $\Gamma_2 = 0.5$), the fidelity of the dissipative gate is 91%.

4.1. Fidelity of a two-qubit gate subject to noise

To introduce how the evolution of a dynamical quantum system is calculated using QuTiP, let us consider a simple system comprised of two qubits that, during a time $T = \pi/4g$, are subject to the coupling Hamiltonian

$$H = g\left(\sigma_x \otimes \sigma_x + \sigma_y \otimes \sigma_y\right), \qquad (17)$$

where g is the coupling strength. Under ideal conditions this coupling realizes the i-SWAP gate between the two qubit states [41, 42]. This can readily be seen by evolving any initial state for the time T, and comparing the final state with the corresponding i-SWAP transformed initial state. We shall assume that the qubits are coupled with their surrounding environments, resulting in qubit energy relaxation and dephasing.

Following the workflow outlined in the previous section, the QuTiP code for this problem can be organized in the following manner. First, define the numerical constants in the problem. For brevity, the code for this step has been omitted. Next, the `Qobj` instances for the Hamiltonian and the initial state may be defined as

```
H = g * (tensor(sigmax(), sigmax()) +
        tensor(sigmay(), sigmay()))

psi0 = tensor(basis(2,1), basis(2,0))
```

To model qubit relaxation and dephasing, we define a list of collapse operators that later will be passed on to the ODE solver. For each qubit we append its associated collapse operator to this list (here called `c_op_list`)

```
sm1 = tensor(sigmam(), qeye(2))
sz1 = tensor(sigmaz(), qeye(2))
c_op_list.append(sqrt(g1 * (1+nth)) * sm1)
c_op_list.append(sqrt(g1 * nth) * sm1.dag())
c_op_list.append(sqrt(g2) * sz1)
```

where the parameter `nth` is the number of environmentally-induced thermal excitations in the steady state. The collapse operators containing σ_- and $\sigma^\dagger$ describe the qubit relaxation and excitation with the rates `g1 * (1+nth)` and `g1 * nth`, respectively, and the collapse operator σ_z models qubit dephasing. These lines of codes are repeated for the second qubit, with the appropriate change in the definition of the operators (i.e., the arguments in the `tensor` function are switched).

At this point we are ready to let QuTiP calculate the time-evolution of the system. In the following example we use the master equation ODE solver `odesolve`. In addition to the Hamiltonian, initial state, and the list of collapse operator, we pass a list `tlist` to the solver that contains the times at which we wish to evaluate the density matrix.

```
tlist     = linspace(0, T, 100)
rho_list  = odesolve(H, psi0, tlist, c_op_list, [])
rho_final = rho_list[-1]
```

If the last parameter is empty, as in this example, all QuTiP time-evolution solvers return the full density matrix (or state vector) corresponding to the times in `tlist`. Alternatively, a list of operators may be passed as last argument to the solver, in which case it will return the corresponding expectation values.

Given the output of density matrices, we may now calculate the corresponding expectation values for selected quantum operators. For example, to calculate the excitation probability of the two qubits as a function of time, we may use the QuTiP function `expect`

```
n1 = expect(sm1.dag() * sm1, rho_list)
n2 = expect(sm2.dag() * sm2, rho_list)
```

Here, `n1` and `n2` are now real *NumPy* arrays of expectation values, suitable for plotting or saving to file.

Finally, to quantify the difference between the lossy i-SWAP gate and its ideal counterpart, we calculate the fidelity `fidelity`

```
U = (-1j * H * pi / (4*g)).expm()
psi_ideal = U * psi0
rho_ideal = psi_ideal * psi_ideal.dag()
f = fidelity(rho_ideal, rho_final).
```

The results are shown in Fig. 6, where the expectation values for the two qubits, as a function of time, is plotted both with and without dissipation. The full code for this example is listed in Appendix B.3.

4.2. Jaynes-Cumming model

The same method used in the previous section can calculate the dynamics of the Jaynes-Cumming model Eq. (2). Only the definitions of the Hamiltonian, initial state, and collapse operators need to be changed to solve this problem. For the Jaynes-Cumming model, the Hamiltonian and a possible initial state were given in Sec. 2, and we need only to define collapse operators before the system can be evolved using one of the QuTiP solvers. The cavity and the atom relaxation rates are κ and Γ, respectively. In this example, only the cavity is coupled to an environment with Boltzmann occupation number n_{th}. We can write the collapse operators for the cavity

```
a = tensor(destroy(N), qeye(2))
c_ops.append(sqrt(kappa * (1+n_th)) * a)
c_ops.append(sqrt(kappa * n_th) * a.dag())
```
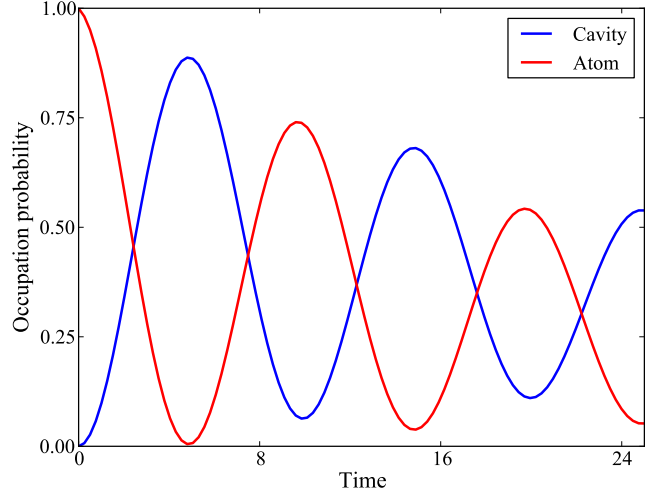

Figure 7: (Color) Evolution of the Jaynes-Cummings Hamiltonian (2) in a thermal environment characterized by $\langle n \rangle_{\text{th}} = 0.75$. Initially, only the atom is excited, but the atom-cavity coupling results in a coherent energy transfer between the two systems, a phenomenon known as vacuum Rabi oscillations. Here, the atom and the cavity are resonant, $\omega_0 = \epsilon = 2\pi$, the coupling strength $g/\omega_0 = 0.05$, and the atom and cavity relaxation rates are $\gamma/(\omega_0/2\pi) = 0.05$ and $\kappa/(\omega_0/2\pi) = 0.005$, respectively.

and for the atom

```
sm = tensor(qeye(N), destroy(2))
c_ops.append(sqrt(gamma)* sm)
```

Instead of having the solver return the state, as in Sec. 4.1, we can request that the expectation values for a list of operators be directly calculated at each time step. In this Jaynes-Cumming problem we are interested in the excitation number of the cavity and the atom, and as such can then define a list of expectation value operators

```
expt_ops = [a.dag() * a, sm.dag() * sm]
```

which may be passed as last argument to, for example, the `odesolve` solver.

```
tlist     = linspace(0, 10, 100)
expt_list = odesolve(H, psi0, tlist, c_ops, expt_ops)
```

The solver then returns a *NumPy* array `expt_list` of expectation values. The result of this calculation is shown in Fig. 7, and the complete code is listed in Appendix B.4.

4.3. Trilinear Hamiltonian

To demonstrate the QuTiP Monte-Carlo (MC) solver, we consider the trilinear Hamiltonian that, in the interaction frame, may be written as

$$H = i\hbar K \left(ab^\dagger c^\dagger - a^\dagger bc \right) \qquad (18)$$

consisting of three harmonic oscillator modes conventionally labeled pump (a), signal (b) and idler (c) respectively, with the frequency relation, $\omega_a = \omega_b + \omega_c$ and coupling constant K. This Hamiltonian is the full quantum generalization of the parametric amplifier [43] describing several quantum optics processes, including frequency conversion

9

[44], the interaction of two-level atoms with a single mode resonant EM field [45], and the modeling of Hawking radiation from a quantized black hole [46]. Here we suppose the pump mode is initially in a coherent state, while the signal and idler modes are in the ground state

$$|\psi(0)\rangle = |\alpha\rangle_a |0\rangle_b |0\rangle_c. \qquad (19)$$

As a system comprised of three harmonic modes, this model readily lends itself to MC simulation since the Hilbert space dimensionality increases exponentially with the number of initial excitations in the system $\langle N(0)\rangle_a = |\alpha|^2$. For example, to accurately model an initial pump mode coherent state with $|\alpha|^2 = 10$ requires ~ 17 states, suggesting a minimum Hilbert space dimensionality of $17^3 = 4913$ for simulating Eq. (18); a value five times larger than what can typically be efficiently calculated using the `odesolve` or `eseries` solvers (see Fig. 3).

In QuTiP, the Hamiltonian (18) may be expressed as $(K=1)$

```
H=1j*(a*b.dag()*c.dag()-a.dag()*b*c),
```

with the destruction operators for the pump, signal, and idler modes, a, b and c respectively, created via the tensor product

```
a=tensor(destroy(N),qeye(N),qeye(N))
b=tensor(qeye(N),destroy(N),qeye(N))
c=tensor(qeye(N),qeye(N),destroy(N)).
```

In addition, we may define number operator expectation values and collapse operators in the same manner as previous examples

```
num0,num1,num2=[a0.dag()*a0,a1.dag()*a1,a2.dag()*a2]
C0,C1,C2=[sqrt(2.0*g0)*a0,sqrt(2.0*g1)*a1,sqrt(2.0*g2)*a2]
```

`mcsolve` takes the same input arguments as `odesolve`, save for an additional argument necessary to specify the number of MC trajectories to simulate

```
ntraj=500
```

In Fig. 8 we plot the expectation values for the three modes of the trilinear Hamiltonian (18), with corresponding damping rates, $\gamma_a = 0.2$, $\gamma_b = 0.8$, $\gamma_c = 0.2$, for the initial state given by Eq. (19) with $\alpha = \sqrt{10}$

```
psi0=tensor(coherent(N,sqrt(10)),basis(N,0),basis(N,0))
avgs=mcsolve(H,psi0,tlist,ntraj,[C0,C1,C2],[num0,num1,num2])
```

Had we not defined any collapse operators, the evolution calculated by `mcsolve` reduces to the Schrödinger equation (5). This evolution is also presented in Fig. (8). The underlying QuTiP code may be found in Appendix B.5.

4.4. Landau-Zener transitions

Landau-Zener transitions [47] are an interesting problem that involves a quantum two-level system with a time-dependent energy splitting. The Hamiltonian is

$$H(t) = \frac{\Delta}{2}\sigma_x + \frac{vt}{2}\sigma_z, \qquad (20)$$

where Δ is the tunneling rate, and v is the rate of change in the bare qubit energy splitting. The Landau-Zener transition theory analytically describes how the final state at
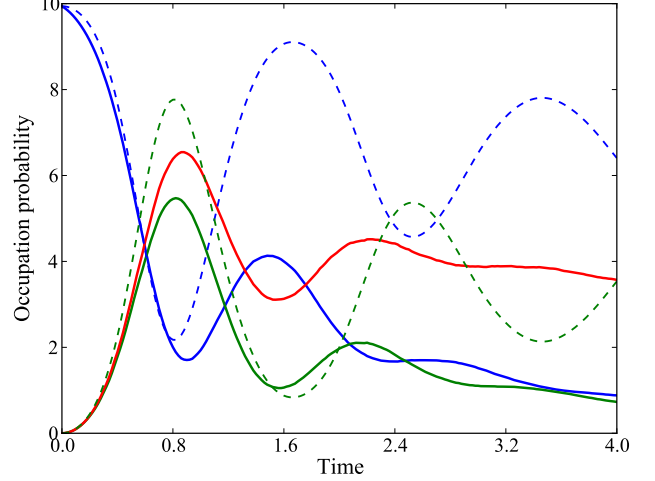

Figure 8: (Color) Occupation numbers for the three modes of the trilinear Hamiltonian (18), averaged over 1000 trajectories, for an initial state Eq. (19) with $\alpha = \sqrt{10}$. In this simulation, the environment is assumed to be at zero temperature. The pump (blue), signal (green), and idler (red) mode damping rates are $\gamma_a = 0.2$, $\gamma_b = 0.8$, and $\gamma_c = 0.2$, respectively. The closed system evolution (dashed-colors) is also presented, where the idler mode is omitted as its evolution is identical to that of the signal.

$t \to \infty$ is related to the initial state at $t \to -\infty$. In particular, the probability of an adiabatic transition from $|1\rangle$ to $|0\rangle$ is given by the Landau-Zener formula

$$P = 1 - \exp\left(-\frac{\pi\Delta^2}{2v}\right). \qquad (21)$$

Using QuTiP, we can easily integrate the system dynamics numerically, and obtain the state of the system for any intermediate value of t.

The Landau-Zener problem differs from the previous examples in that the Hamiltonian is explicitly time-dependent. The QuTiP solvers `odesolve` and `mcsolve` both support time-dependent Hamiltonians. In order to specify an arbitrary time-dependent Hamiltonian, a callback function may be passed as the first argument to the time-evolution solvers (in place of the `Qobj` instance that normally represents the Hamiltonian). The callback function is expected to return the value of the Hamiltonian at the given point in time `t`, which is the first argument to the callback function.

```
def hamiltonian_t(t, args):
    H0 = args[0]
    H1 = args[1]
    return H0 + t * H1
```

In addition, a second argument passed to the callback function is a user-defined list of parameters. For performance reasons, it is appropriate to let this list contain pre-calculated `Qobj` instances for the constant parts of the Hamiltonian. For the Landau-Zener problem this corresponds to

```
H0 = delta/2.0 * sigmax()
H1 =     v/2.0 * sigmaz()
H_args = (H0, H1)
```

The list of arguments for the Hamiltonian callback function is then passed on to the time-evolution solver (as the
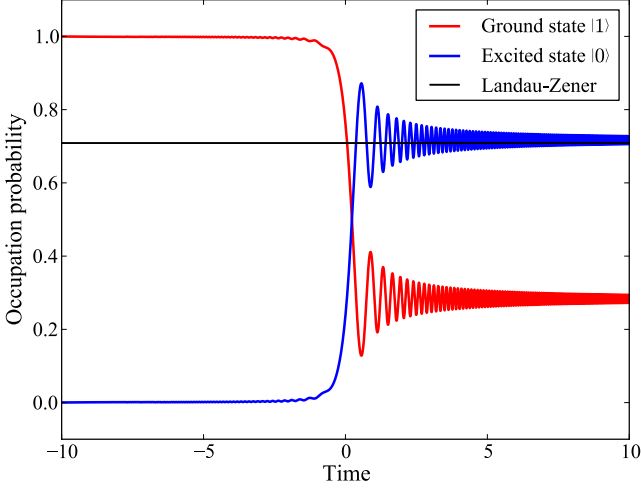
Figure 9: (Color) The occupation probability of the $|1\rangle$ (red curve) and $|0\rangle$ (blue curve) states of a quantum two-level system throughout a Landau-Zener transition. The solid black line is the final state according to the Landau-Zener formula Eq. (21). The parameters used in this calculation are $\Delta = 0.5 \times 2\pi$ and $v = 2.0 \times 2\pi$.
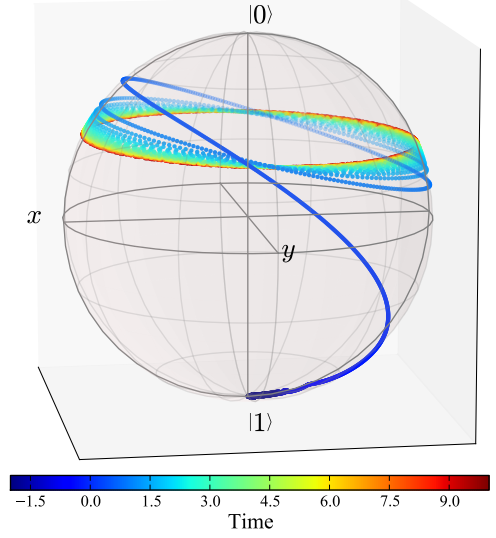

Figure 10: (Color) Bloch sphere representation of the Landau-Zener transition presented in Fig. 9. Here, the color of the data points corresponds to the evolution time.

very last argument), along with the callback function itself (as first argument).

```
expt_list = odesolve(hamiltonian_t, psi0, tlist,
                     c_op_list, expt_list, H_args)
```

The result of this calculation is shown in Fig. (9), which shows the intermediate dynamics of the system in terms of the occupation probabilities of the $|0\rangle$ and $|1\rangle$ states. The full code is shown in Appendix B.6. Adding the operators `sigmax, sigmay` and `sigmaz` to `expt_list`, this evolution can also be visualized on the Bloch sphere using QuTiP's built in `Bloch` class, as demonstrated in Fig. 10. The QuTiP code for this figure is given in Appendix B.7.

Although simple, this example illustrates the support for time-dependent Hamiltonians in QuTiP. It is a straightforward exercise to implement an arbitrary time-dependence by changing the definition of the Hamiltonian callback function, or by modifying the time-independent part of the Hamiltonian to correspond to a more complicated quantum system.

5. Performance

As with any scientific simulation, the performance of the underlying numerical routines in QuTiP is an important factor to consider when choosing which software to implement in the analysis of the problem at hand. In simulating quantum dynamics on a classical computer, this is especially important given that creating composite systems using the tensor product leads to an exponential increase in the total Hilbert space dimensionality. Thus, it is beneficial to compare the performance of QuTiP to the other currently available quantum simulation packages. In this section we compare the simulation times of the master

equation and Monte-Carlo solvers in QuTiP, to the those of the qotoolbox, as a function of Hilbert space size.

To compare the QuTiP master equation solver, `odesolve`, to the qotoolbox equivalent (also called `odesolve`), we evaluate the coupled oscillator equation ($\hbar = 1$)

$$H = \omega_a a^\dagger a + \omega_b b^\dagger b + \omega_{ab}\left(a^\dagger b + a b^\dagger\right), \qquad (22)$$

with $\omega_a = \omega_b = 2\pi$ and $\omega_{ab} = 0.1 \times 2\pi$. In addition, we consider the situation in which one resonator is damped with a corresponding dissipation rate $g = 0.05$. Here, the initial state of the system is the tensor product of Fock states $|\psi(0)\rangle = |N\rangle_a|N - 1\rangle_b$, where N is the number of states in the truncated Hilbert space for each oscillator. The total time needed to simulate the dynamics over the time range $t \in [0, 10]$ is shown in Fig. (11). We see that the QuTiP solver easily outperforms the qotoolbox as the Hilbert space dimensionality $D = N^2$ increases. For the largest dimensionality considered in Fig. (11) $D = 200$, QuTiP is ~ 4 times faster than the qotoolbox single-precision C-code implementation, even though QuTiP is performing double-precision calculations with a small Python overhead.

The trilinear Hamiltonian model from Sec. 4.3 provides a useful demonstration of the multiprocessing routines used by the QuTiP `mcsolve` function. Here, the independent MC trajectories are run in parallel, with the number of simultaneous trajectories determined by the number of processing cores. For the large Hilbert spaces associated with the trilinear Hamiltonian, the increased overhead generated from multiprocessing is overcome by the gains in running Monte-Carlo trajectories in parallel. In Fig. (12), we highlight these performance gains in simulating Eq. (8) by plotting the computation time over a
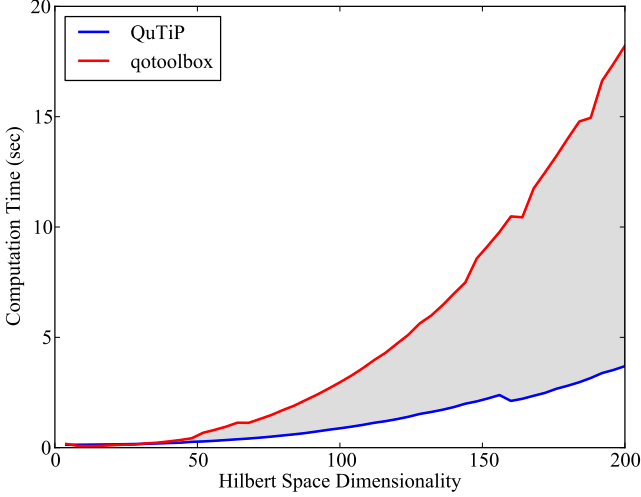
11

Figure 11: (Color) Comparison of computation times, averaged over three trials, for solving Eq. (22) in both QuTiP (blue) and the qotoolbox (red) as a function of Hilbert space dimension. The shaded region highlights the increasing performance benefit from using the QuTiP solver as the dimensionality increases. Simulations were performed on a quad-core 2.8 Ghz processor.
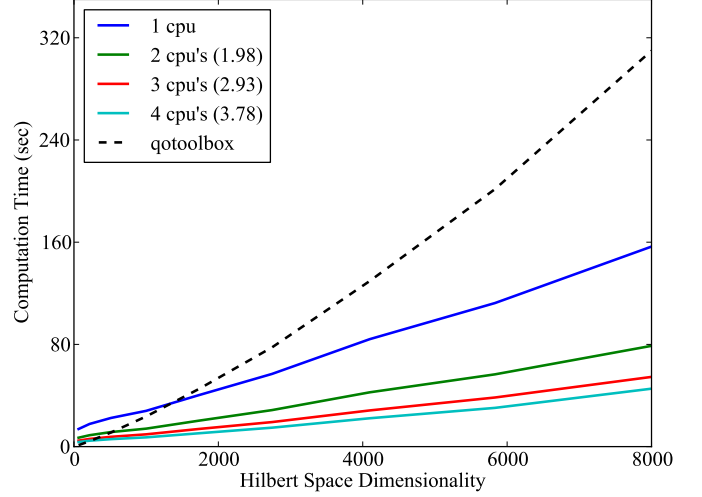

Figure 12: (Color) Comparison of computation times, averaged over three runs, between QuTiP and the qotoolbox (dashed) in simulating the trilinear Hamiltonian from Eq.(18) for an initial pump mode coherent state with expectation value $\langle N \rangle_a = \sqrt[3]{D}/4$ as a function of the Hilbert space dimensionality D. The average performance enhancement from using multiple processors, as compared to the single-processor performance, is given in the legend. Ideally, this value should be equal to the number of processors. Computations were performed on a quad-core 2.8 Ghz processor.

range of Hilbert space dimensions. For comparison, we also plot the times required for identical simulations using the qotoolbox, which is limited to a single processor. For this simulation, the qotoolbox outperforms our QuTiP implementation for system sizes $D \lesssim 500$, even when multiple processors are utilized. This is due to the Python overhead needed to implement the Monte-Carlo algorithm discussed in Sec. 3.2. However, as shown in Fig. (3), using the master equation is more appropriate for systems of this size. As with the master equation solver, Fig. (11), the benefits of using the QuTiP Monte-Carlo solver become appreciable as the system size increases. Even for a single-processor, the QuTiP `mcsolve` routine outperforms the qotoolbox after $D \approx 1500$, where the Python overhead is no longer a limiting factor. When using multiple processing cores, the Monte-Carlo solver performance gain nearly equals the number of processors available, the ideal situation.

6. Conclusion

We have presented a new, open-source framework for the numerical simulation of open quantum systems implemented in the Python programming language. This framework is suitable for a wide range of computational problems in quantum systems, including unitary and dissipative time-evolution, spectral and steady-state properties, as well as advanced visualization techniques. In this work we have described the basic structure of the framework, the `Qobj` class, and the primary evolution solvers, `odesolve` and `mcsolve`. In addition, we have highlighted a number of examples intended to give the reader a flavor of the types of problems for which QuTiP is suitable. For

more in-depth documentation, and more elaborate examples using the functions listed in Table A.2, we refer the reader to the QuTiP website [23]. There, one may download the latest version of this framework, as well as find installation instructions for the most common computer platforms. The version of the framework described in this paper is QuTiP 1.1.3.

As with any initial software release, the performance of QuTiP can likely be improved in the future, with additional portions of the code being optimized with Cython [48] or implemented in PyOpenCL [49].

Appendix A. QuTiP function list

States	
basis / fock	Creates single basis (Fock) state in Hilbert space.
coherent	Single-mode coherent state with complex amplitude α.
coherent_dm	Coherent state density matrix with complex amplitude α.
fock_dm	Density matrix representation of a single basis (Fock) state.
qstate	Tensor product state for any number of qubits in either the ground or excited states.
thermal_dm	Thermal state density matrix.

Operators	
qeye	Identity operator.
create	Bosonic creation operator.
destroy	Bosonic annihilation operator.
displace	Single-mode displacement operator.
squeez	Single-mode squeezing operator.
num	Number operator.
sigmax	Pauli spin-1/2 σ_x operator.
sigmay	Pauli spin-1/2 σ_y operator.
sigmaz	Pauli spin-1/2 σ_z operator.
sigmap	σ_+ operator.
sigmam	σ_- operator.
jmat	Spin-j operator.

Functions on states	
entropy_vn	Von-Neumann entropy of a density matrix.
expect	Calculates the expectation value of an operator.
fidelity	Calculates the fidelity between two density matrices.
ket2dm	Converts a ket vector to a density matrix.
liouvillian	Assembles the Liouvillian super-operator from a Hamiltonian and a list of collapse operators.
orbital	Calculates an angular wave function on a sphere.
ptrace	Partial trace of composite quantum object.
qfunc	Husimi-Q function of a given state vector or density matrix.
simdiag	Simultaneous diagonalization of commuting Hermitian operators.
tensor	Calculates tensor product from list of input operators or states.
tidyup	Removes small elements from a quantum object.
tracedist	Trace distance between two density matrices.
wigner	Wigner function of a given state vector or density matrix.

Evolution	
correlation_es	Two-time correlation function using exponential series.
correlation_mc	Two-time correlation function using Monte-Carlo method.
correlation_ode	Two-time correlation function using ODE solver.
correlation_ss_es	Two-time correlation function using quantum regression theorem.
essolve	State or density matrix evolution using exponential series expansion of ODE.
mcsolve	Stochastic Monte-Carlo wave function solver.
Odeoptions	Options class for ODE integrators used by mcsolve and odesolve.
odesolve	ODE solver for density matrix evolution.
propagator	Calculates the propagator $U(t)$ for a density matrix or wave function.
propagator_steadystate	Steady state for successive applications of the propagator $U(t)$.
steadystate	Calculates the steady state for the supplied Hamiltonian.

Utilities	
about	Information on installed version of QuTiP and its dependencies.
Bloch	Class for plotting vectors and data points on the Bloch sphere.
clebsch	Calculates a Clebsch-Gordon coefficient.
demos	Runs built-in demos scripts.
eseries	Exponential series representation of a time-dependent quantum object.
ode2es	Exponential series describing the time evolution of an initial state.
parfor	Parallel execution of a for-loop over a single-variable.
Qobj	Class for creating user-defined quantum objects.
sphereplot	Plots an array of values on a sphere.

Table A.2: List of user-accessible functions in QuTiP. Additional information about each function may be obtained by calling '*function_name*?' from the Python command-line, or by going to the QuTiP website [23].

Appendix B. QuTiP codes

In this section we display the QuTiP codes underlying the calculations performed in generating Figures 2, 4, 6, 7, 8, 9, and 10. For brevity, the code segments associated with plotting have been omitted. The codes, in their entirety, may be viewed at the QuTiP website [23].

Appendix B.1. Figure 2: Non-RWA Jaynes-Cummings Model

```
from qutip import *
## set up the calculation ##
wc = 1.0 * 2 * pi # cavity frequency
wa = 1.0 * 2 * pi # atom frequency
N = 20           # number of cavity states
g = linspace(0, 2.5, 50)*2*pi # coupling strength vector
## create operators ##
a  = tensor(destroy(N), qeye(2))
sm = tensor(qeye(N), destroy(2))
nc = a.dag() * a
na = sm.dag() * sm
## initialize output arrays ##
na_expt = zeros(len(g))
nc_expt = zeros(len(g))
## run calculation ##
for k in range(len(g)):
    ## recalculate the hamiltonian for each value of g ##
    H = wc*nc+wa*na+g[k]*(a.dag()+a)*(sm+sm.dag())
    ## find the groundstate ##
    ekets, evals = H.eigenstates()
    psi_gnd = ekets[0]
    ## expectation values ##
    na_expt[k] = expect(na, psi_gnd) # qubit occupation
    nc_expt[k] = expect(nc, psi_gnd) # cavity occupation
## Calculate Wigner function for coupling g=2.5 ##
rho_cavity = ptrace(psi_gnd,0) # trace out qubit
xvec = linspace(-7.5,7.5,200)
## Wigner function ##
W = wigner(rho_cavity, xvec, xvec)
```

Appendix B.2. Figure 4: Monte-Carlo relaxation in a thermal environment

```
from qutip import *
N=5             # number of basis states to consider
a=destroy(N)    # cavity destruction operator
H=a.dag()*a     # harmonic oscillator Hamiltonian
psi0=basis(N,1) # initial Fock state with one photon
kappa=1.0/0.129 # coupling to heat bath
nth= 0.063      # temperature with <n>=0.063
## collapse operators ##
c_op_list = []
## decay operator ##
c_op_list.append(sqrt(kappa * (1 + nth)) * a)
## excitation operator ##
c_op_list.append(sqrt(kappa * nth) * a.dag())
## run simulation ##
ntraj=904 # number of MC trajectories
tlist=linspace(0,0.6,100)
mc = mcsolve(H,psi0,tlist,ntraj,c_op_list, [])
me = odesolve(H,psi0,tlist,c_op_list, [a.dag()*a])
## expectation values ##
ex1=expect(num(N),mc[0])
ex5=sum([expect(num(N),mc[k]) for k in range(5)],0)/5
ex15=sum([expect(num(N),mc[k]) for k in range(15)],0)/15
ex904=sum([expect(num(N),mc[k]) for k in range(904)],0)/904
```

Appendix B.3. Figure 6: Dissipative i-SWAP gate

```
from qutip import *
g  = 1.0 * 2 * pi  # coupling strength
g1 = 0.75          # relaxation rate
g2 = 0.05          # dephasing rate
n_th = 0.75        # bath temperature
T = pi/(4*g)
H = g * (tensor(sigmax(), sigmax()) +
         tensor(sigmay(), sigmay()))
psi0 = tensor(basis(2,1), basis(2,0))
c_op_list = []
## qubit 1 collapse operators ##
sm1 = tensor(sigmam(), qeye(2))
sz1 = tensor(sigmaz(), qeye(2))
c_op_list.append(sqrt(g1 * (1+n_th)) * sm1)
c_op_list.append(sqrt(g1 * n_th) * sm1.dag())
c_op_list.append(sqrt(g2) * sz1)
## qubit 2 collapse operators ##
sm2 = tensor(qeye(2), sigmam())
sz2 = tensor(qeye(2), sigmaz())
c_op_list.append(sqrt(g1 * (1+n_th)) * sm2)
c_op_list.append(sqrt(g1 * n_th) * sm2.dag())
c_op_list.append(sqrt(g2) * sz2)
## evolve the system ##
tlist = linspace(0, T, 100)
rho_list  = odesolve(H, psi0, tlist, c_op_list, [])
rho_final = rho_list[-1]
## calculate expectation values ##
n1 = expect(sm1.dag() * sm1, rho_list)
n2 = expect(sm2.dag() * sm2, rho_list)
## calculate the fidelity ##
U = (-1j * H * pi / (4*g)).expm()
psi_ideal = U * psi0
rho_ideal = psi_ideal * psi_ideal.dag()
f = fidelity(rho_ideal, rho_final)
```

Appendix B.4. Figure 7: Dissipative Jaynes-Cumming model

```
from qutip import *
N = 5                     # number of cavity states
omega0 = epsilon = 2 * pi # frequencies
g = 0.05 * 2 * pi         # coupling strength
kappa = 0.005             # cavity relaxation rate
gamma = 0.05              # atom relaxation rate
n_th = 0.75              # bath temperature
## Hamiltonian and initial state ##
a  = tensor(destroy(N), qeye(2))
sm = tensor(qeye(N), destroy(2))
sz = tensor(qeye(N), sigmaz())
H  = omega0 * a.dag() * a + 0.5 * epsilon * sz
     + g * (a.dag() * sm + a * sm.dag())
psi0 = tensor(fock(N,0), fock(2,1)) # excited atom
## Collapse operators ##
c_ops = []
c_ops.append(sqrt(kappa * (1+n_th)) * a)
c_ops.append(sqrt(kappa * n_th) * a.dag())
c_ops.append(sqrt(gamma) * sm)
## Operator list for expectation values ##
expt_ops = [a.dag() * a, sm.dag() * sm]
## Evolution of the system ##
tlist = linspace(0, 10, 100)
expt_data = odesolve(H, psi0, tlist, c_ops, expt_ops)
```

Appendix B.5. Figure 8: Trilinear Hamiltonian

```
from qutip import *
N=17 # number of states for each mode
## damping rates ##
g0=g2=0.1
g1=0.4
```

```
alpha=sqrt(10) # initial coherent state alpha
tlist=linspace(0,4,201) # list of times
ntraj=1000#number of trajectories
## lowering operators ##
a0=tensor(destroy(N),qeye(N),qeye(N))
a1=tensor(qeye(N),destroy(N),qeye(N))
a2=tensor(qeye(N),qeye(N),destroy(N))
## number operators ##
n0,n1,n2=[a0.dag()*a0,a1.dag()*a1,a2.dag()*a2]
## dissipative operators ##
C0,C1,C2=[sqrt(2.0*g0)*a0,sqrt(2.0*g1)*a1,sqrt(2.0*g2)*a2]
## initial state ##
psi0=tensor(coherent(N,alpha),basis(N,0),basis(N,0))
## trilinear Hamiltonian ##
H=1j*(a0*a1.dag()*a2.dag()-a0.dag()*a1*a2)
## run Monte-Carlo ##
avgs=mcsolve(H,psi0,tlist,ntraj,[C0,C1,C2],[n0,n1,n2])
## run Schrodinger ##
reals=mcsolve(H,psi0,tlist,1,[],[n0,n1,n2])
```

Appendix B.6. Figure 9: Landau-Zener transitions
```
from qutip import *
## callback function for time-dependence ##
def hamiltonian_t(t, args):
    H0 = args[0]
    H1 = args[1]
    return H0 + t * H1
delta = 0.5 * 2 * pi
v = 2.0 * 2 * pi # sweep rate
## arguments for Hamiltonian ##
H0 = delta/2.0 * sigmax()
H1 = v/2.0 * sigmaz()
H_args = (H0, H1)
psi0 = basis(2,0)
## expectation operators ##
sm = destroy(2)
sx=sigmax();sy=sigmay();sz=sigmaz()
expt_op_list = [sm.dag() * sm,sx,sy,sz]
## evolve the system ##
tlist = linspace(-10.0, 10.0, 1500)
expt_list = odesolve(hamiltonian_t, psi0, tlist,
                     [], expt_op_list, H_args)
```

Appendix B.7. Figure 10: Bloch sphere representation of Landau-Zener transition

Following the code from Appendix B.6:

```
import matplotlib as mpl
from matplotlib import cm
## create Bloch sphere instance ##
b=Bloch()
## normalize colors to times in tlist ##
nrm=mpl.colors.Normalize(-2,10)
colors=cm.jet(nrm(tlist))
## add data points from expectation values ##
b.add_points([p_ex[1],p_ex[2],-p_ex[3]],'m')
## customize sphere properties ##
b.point_color=list(colors)
b.point_marker=['o']
b.point_size=[8]
b.view=[-9,11]
b.zlpos=[1.1,-1.2]
b.zlabel=['$\left|0\right>_{f}$','$\left|1\right>_{f}$']
## plot sphere ##
b.show()
```

References

[1] H.-P. Breuer, F. Petruccione, The Theory of Open Quantum Systems, Oxford University, 2002.

[2] G. Lindblad, Commun. Math. Phys. 48 (1976) 119.
[3] G. Z. K. Horvath, P. L. Knight, R. C. Thompson, Contemp. Phys. 38 (1997) 25.
[4] M. B. Plenio, P. L. Knight, Rev. Mod. Phys. 70 (1998) 101.
[5] I. Buluta, S. Ashhab, F. Nori, Reports on Progress in Physics 74 (2011) 104401.
[6] R. P. Feynman, Int. J. Theor. Phys. 21 (1982) 467.
[7] I. Buluta, F. Nori, Science 326 (2009) 108.
[8] S. Haroche, J.-M. Raimond, Exploring the Quantum: Atoms, Cavities, and Photons, Oxford University, 2006.
[9] J. L. O'Brien, A. Furusawa, J. Vučkovic, Nat. Photonics 3 (2009) 687.
[10] R. Blatt, D. Wineland, Nature 453 (2008) 1008.
[11] J. Q. You, F. Nori, Phys. Today 58 (2005) 42.
[12] R. J. Schoelkopf, S. M. Girvin, Nature 451 (2008) 664.
[13] J. Q. You, F. Nori, Nature 474 (2011) 589.
[14] A. D. Armour, M. P. Blencowe, New J. Phys. 10 (2008) 095004.
[15] M. P. Blencowe, A. D. Armour, New J. Phys. 10 (2008) 095005.
[16] A. D. O'Connell, M. Hofheinz, M. Ansmann, R. C. Bialczak, M. Lenander, E. Lucero, M. Neeley, D. Sank, H. Wang, M. Weides, J. Wenner, J. M. Martinis, A. N. Cleland, Nature 464 (2010) 697.
[17] J. D. Teufel, T. Donner, L. Dale, J. W. Harlow, M. S. Allman, K. Cicak, A. J. Sirois, J. D. Whittaker, K. W. Lehnert, R. W. Simmonds, Nature 475 (2011) 359.
[18] R. Schack, T. A. Brun, Comput. Phys. Commun. 102 (1997) 210.
[19] S. M. Tan, J. Opt. B: Quantum Semiclass. Opt. 1 (1999) 424.
[20] A. Vukics, H. Ritsch, Eur. Phys. J. D 44 (2007) 585.
[21] The Mathworks Inc., Matlab, http://www.mathworks.com/, 2011.
[22] G. van Rossum, et al., The Python progamming language, http://www.python.org/, 2011.
[23] P. D. Nation, J. R. Johansson, QuTiP: Quantum Toolbox in Python, http://code.google.com/p/qutip/, 2011.
[24] T. E. Oliphant, Computing in Science & Engineering 9 (2007) 10.
[25] E. Jones, T. Oliphant, P. Peterson, et al., SciPy: Open source scientific tools for Python, http://www.scipy.org/, 2011.
[26] J. D. Hunter, Computing in Science & Engineering 9 (2007) 90.
[27] M. A. Nielsen, I. L. Chuang, Quantum Computation and Quantum Information, Cambridge, 2000.
[28] J. Casanova, G. Romero, I. Lizuain, J. J. García-Ripoll, E. Solano, Phys. Rev. Lett. 105 (2010) 263603.
[29] S. Ashhab, F. Nori, Phys. Rev. A 81 (2010) 042311.
[30] X. Cao, J. Q. You, H. Zheng, F. Nori, New J. Phys. 13 (2011) 073002.
[31] P. Forn-Díaz, J. Lisenfield, D. Marcos, García-Ripoll, E. Solano, C. J. P. M. Harmans, J. E. Mooij, Phys. Rev. Lett. 105 (2010) 237001.
[32] P. Nataf, C. Ciuti, Phys. Rev. Lett. 104 (2010) 023601.
[33] C. W. Gardiner, P. Zoller, Quantum Noise, Springer, 3rd edition, 2004.
[34] D. F. Walls, G. J. Milburn, Quantum Optics, Springer, 2nd edition, 2008.
[35] C. Guerlin, J. Bernu, S. Deléglise, C. Sayrin, S. Gleyzes, S. Kuhr, M. Brune, J.-M. Raimond, S. Haroche, Nature 448 (2007) 889.
[36] A. N. Vamivakas, C.-Y. Lu, C. Matthisen, Y. Zhao, S. Fält, A. Badolato, M. Atatüre, Nature 467 (2010) 297.
[37] J. Dalibard, Y. Castin, K. Mølmer, Phys. Rev. Lett. 68 (1992) 580.
[38] R. Dum, P. Zoller, H. Ritsch, Phys. Rev. A 45 (1992) 4879.
[39] K. Mølmer, Y. Castin, J. Dalibard, J. Opt. Soc. Am. B 10 (1993) 524.
[40] S. Gleyzes, S. Kuhr, C. Guerlin, J. Bernu, S. Deléglise, U. Busk Hoff, M. Brune, J.-M. Raimond, S. Haroche, Nature 446 (2007) 297.
[41] N. Schuch, J. Siewert, Phys. Rev. A 67 (2003) 032301.
[42] M. Steffen, M. Ansmann, R. C. Bialczak, N. Katz, E. Lucero, R. McDermott, M. Neeley, E. M. Weig, A. N. Cleland, J. M.

Martinis, Science 313 (2006) 1423–1425.

[43] B. R. Mollow, R. J. Glauber, Phys. Rev. 160 (1967) 1076.

[44] D. F. Walls, R. Barakat, Phys. Rev. A 1 (1970) 446.

[45] M. Tavis, F. W. Cummings, Phys. Rev. 170 (1968) 379.

[46] P. D. Nation, M. P. Blencowe, New J. Phys. 12 (2010) 095013.

[47] S. N. Shevchenko, S. Ashhab, F. Nori, Phy. Rep. 492 (2010) 1.

[48] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn, K. Smith, Computing in Science & Engineering 13 (2011) 31.

[49] A. Klöckner, PyOpenCL, http://mathema.tician.de/software/pyopencl, 2011.