Computer Physics Communications 184 (2013) 1234–1240

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

QuTiP 2: A Python framework for the dynamics of open quantum systems☆

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

[a] Advanced Science Institute, RIKEN, Wako-shi, Saitama 351-0198, Japan
[b] Department of Physics, Korea University, Seoul 136-713, Republic of Korea
[c] Department of Physics, University of Michigan, Ann Arbor, MI 48109-1040, USA

ARTICLE INFO

Article history:
Received 24 October 2012
Received in revised form
28 November 2012
Accepted 29 November 2012
Available online 13 December 2012

Keywords:
Open quantum systems
Lindblad
Bloch–Redfield
Floquet–Markov
Master equation
Quantum Monte Carlo
Python

ABSTRACT

We present version 2 of QuTiP, the Quantum Toolbox in Python. Compared to the preceding version [J.R. Johansson, P.D. Nation, F. Nori, Comput. Phys. Commun. 183 (2012) 1760.], we have introduced numerous new features, enhanced performance, and made changes in the Application Programming Interface (API) for improved functionality and consistency within the package, as well as increased compatibility with existing conventions used in other scientific software packages for Python. The most significant new features include efficient solvers for arbitrary time-dependent Hamiltonians and collapse operators, support for the Floquet formalism, and new solvers for Bloch–Redfield and Floquet–Markov master equations. Here we introduce these new features, demonstrate their use, and give a summary of the important backward-incompatible API changes introduced in this version.

Program Summary

Program title: QuTiP: The Quantum Toolbox in Python
Catalog identifier: AEMB_v2_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEMB_v2_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: GNU General Public License, version 3
No. of lines in distributed program, including test data, etc.: 33625
No. of bytes in distributed program, including test data, etc.: 410064
Distribution format: tar.gz
Programming language: Python.
Computer: i386, x86-64.
Operating system: Linux, Mac OSX.
*RAM:*2+ Gigabytes
Classification: 7.
External routines: NumPy, SciPy, Matplotlib, Cython
Catalog identifier of previous version: AEMB_v1_0
Journal reference of previous version: Comput. Phys. Comm. 183 (2012) 1760
Does the new version supercede the previous version?: Yes
Nature of problem: Dynamics of open quantum systems
Solution method: Numerical solutions to Lindblad, Floquet–Markov, and Bloch–Redfield master equations, as well as the Monte Carlo wave function method.
Reasons for new version: Compared to the preceding version we have introduced numerous new features, enhanced performance, and made changes in the Application Programming Interface (API) for improved functionality and consistency within the package, as well as increased compatibility with existing conventions used in other scientific software packages for Python. The most significant new features include efficient solvers for arbitrary time-dependent Hamiltonians and collapse operators, support for the Floquet formalism, and new solvers for Bloch–Redfield and Floquet–Markov master equations.
Restrictions: Problems must meet the criteria for using the master equation in Lindblad, Floquet–Markov, or Bloch–Redfield form.
Running time: A few seconds up to several tens of hours, depending on size of the underlying Hilbert space.

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding authors.
E-mail addresses: robert@riken.jp (J.R. Johansson), pnation@korea.ac.kr (P.D. Nation).

[1] These authors contributed equally to this work.

0010-4655/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2012.11.019

1. Introduction

The Quantum Toolbox in Python (QuTiP) is a generic framework for numerical simulation and computation of the dynamics of both open and closed quantum systems. This framework follows an object-oriented design that makes the programming of a quantum mechanical problem an intuitive process that closely follows the corresponding mathematical formulation. Although implemented in the interpreted programming language Python, the use of the NumPy and SciPy scientific libraries, and selective optimization using Cython, allows QuTiP to achieve performance that matches, or, in many cases, exceeds that of natively compiled alternatives. At the same time, QuTiP provides a significantly more convenient and flexible programming environment that is easy to learn, and is well suited for use in the classroom.

Since the first major-version release of QuTiP [1] (the 1.x series), active development of the framework has resulted in a significant number of new features and performance enhancements, that have culminated in the second major-version release. Here, we briefly describe the most significant changes and additional functionality introduced in this most recent release.

This paper is organized as follows. In Section 2, we highlight the important API changes introduced in going from QuTiP 1.x to versions 2.x and higher. Section 3 details the primary new features included in this latest version. To illustrate the new functionality in QuTiP, Section 4 contains a selection of examples that highlight how these functions are used in numerical quantum simulations. Finally, a list of all new user-accessible functions, including a brief description, is given in the Appendix.

2. API changes

Here, we list the backward-incompatible changes in the API of QuTiP 2 as compared to the previous version (1.1.4) described in Ref. [1]. These changes are important when porting applications and simulations that are developed for QuTiP 1.1.4 to QuTiP 2.0 and higher. For newly developed simulations, we recommend following the documentation and examples for QuTiP 2.1 [2], in which case the following API changes are not relevant. Note that this article covers version 2.1.0 of the QuTiP framework, and incorporates several features added since the initial 2.0 release.

1. All quantum dynamics solvers (`mcsolve`, `mesolve`, `essolve`, `brmesolve`, and `fmmesolve`) now return an `Odedata` instance, that contains all information about the solution (as opposed to data lists or Qobjs lists as in QuTiP version 1.1.4). A typical call to a time-evolution solver using the new API is

   ```
   sol = solver(H, psi0, tlist, c_ops, e_ops)
   ```

   where the return object `sol` has the attribute `sol.expect` or `sol.states` containing the lists of expectation values or Qobj instances, respectively, that would be returned by the same solver in QuTiP 1.1.4. With this new API, each solver can optionally store additional information in the return object such as, for example, the collapse times calculated in the Monte Carlo solver.
2. The name of the function for the Lindblad master equation solver has been changed from `odesolve` to `mesolve`. The `odesolve` function can still be called; however, it is officially deprecated, and will be removed in a future release. Being a QuTiP version 1.x function, `odesolve` does not return an `Odedata` object.
3. The order of the return values of the method `Qobj.eigenstates` have been swapped, so that the eigenenergies and eigenstates of a `Qobj` instance `op` are now returned in the following order:

   ```
   eigvals, eigkets = op.eigenstates()
   ```

4. Functions for calculating correlations using different solvers have now been consolidated under the functions `correlation` and `correlation_ss`, for transient and steady-state correlations, respectively. Here, the selection of the underlying dynamics solver now is specified using the optional keyword argument `solver` that defaults to the Lindblad master equation (`mesolve`) if it is not explicitly specified. For example,

   ```
   corr_mat = correlation(H, rho0, tlist, taulist,
                          c_op, A, B, solver="me")
   ```

   where `solver` can be `"me"` or `"es"`.

3. New features

QuTiP 2 includes a wide variety of new computational functions, as well as utility functions for better handling of data. Here, we give a brief description the new major features in QuTiP 2.1. For full documentation of these new features, as well as the rest of the QuTiP package, see the QuTiP 2.1 Documentation [2]. Examples illustrating the usage of these functions can be found in Section 4.

- **Support for time-dependent collapse operators**: We have created a new system for representing time-dependent quantum operators used in defining system Hamiltonians and collapse operators for the Lindblad master equation and the Monte Carlo solvers. This allows support for arbitrary time-dependent collapse operators (the new method is still backwards compatible with the Python function callback method used for time-dependent Hamiltonians in QuTiP version 1.1.4). This new method of defining arbitrary time dependencies is both more efficient and more flexible, allowing for high-performance simulations of arbitrary time-dependent quantum systems. In particular, many problems of interest may be compiled at runtime into C code via Cython [3]. This particular feature, and its implementation, will be discussed elsewhere [4].
- **Floquet formalism, Floquet–Markov master equation**: For periodic time-dependent systems, the Floquet formalism can be a useful technique where the original time-dependent problem is transformed into a time-independent problem using the time-dependent Floquet modes as the basis set. In QuTiP 2.0 we added a new module for the Floquet-related decomposition of time-dependent problems, and the evolution of unitary and dissipative dynamics using equations of motion and master equations in the Floquet formalism.
- **Bloch–Redfield master equation solver:** A new quantum dynamics solver for the time evolution according to the Bloch–Redfield master equation is now included in QuTiP. While not as efficient as the Lindblad master equation solver, in situations where the environment is expressed in terms of its noise power spectrum, rather than phenomenological decay and dephasing rates used in the Lindblad formalism, the Bloch–Redfield master equation has significant advantages.
- **Quantum process tomography:** Quantum process tomography (QPT) [5] is a useful technique for characterizing experimental implementations of quantum gates involving a small number of qubits. It can also be a useful theoretical tool that gives insight into how a given process transforms density matrices, and it can be used, for example, to study how noise or other imperfections deteriorate quantum gate operations. Unlike the fidelity or trace distance, that give a single number indicating how far from ideal a gate is, quantum process tomography gives detailed information as to exactly what kinds of error various imperfections and losses introduce.
- **Functions for generating random states and matrices**: It is now possible to generate random kets, density matrices, Hamiltonians, and Unitary operators. This includes the ability to set the sparsity (density) of the resultant quantum object.

- **Support for sparse eigensolvers**: The quantum object (Qobj) methods `eigenstates`, `eigenenergies`, and `groundstate` can now use sparse eigensolvers for systems with large Hilbert spaces. This option is not enabled by default, and must be set with the keyword argument `sparse`.
- **New entropy and entanglement functions**: Functions for calculating the concurrence, mutual information, and conditional entropy have been added.
- **New operator norms**: When calculating operator norms, one can now select between the following norms: trace, Frobius, one, and max. The trace norm is chosen by default. For ket and bra vectors, only the L2-norm is available.
- **Saving and loading data**: Saving and loading of quantum objects and array data is now internally supported by QuTiP. The storage and retrieval of all quantum objects and Odedata objects can be accomplished via the `qsave` and `qload` functions, respectively. In order to facilitate the export of QuTiP data to other programs, `file_data_store` and `file_data_read` allow the user to read and write array data, both real and complex, into text files with a wide variety of formatting options.
- **Performance improvements**: In QuTiP 2.1, numerous performance optimizations have been implemented, including more efficient quantum object creation, significantly faster `ptrace` implementation, and an improved `steadystate` solver.
- **Unit tests for verification of installation**: The installation of QuTiP 2.1 comes with a set of unit tests that can be used to verify that the installation was successful, and that the underlying routines are functioning as expected.

4. Example scripts featuring new functionality

In this section, we highlight, via examples, several of the main features added in QuTiP 2.1 and listed in Section 3. Although we will demonstrate the use of the new time-dependent evolution framework, a full discussion of this feature is presented elsewhere [4]. The examples listed below, as well as a growing collection of additional demonstrations, can be found on the QuTiP website [2], or run after installing QuTiP using the `demos` function. For brevity, we do not include the portions of code related to figure generation using the matplotlib framework [6].

4.1. API changes to dynamics solvers

Here we demonstrate using the new `Odedata` class that is returned by the `mcsolve`, `mesolve`, `brmesolve`, and `fmmesolve` evolution solvers in QuTiP version 2.1. To better illustrate the API changes, we have recoded the two-qubit gate example from Ref. [1] Sec. (4.1) that is written using the older QuTiP 1.x API. The sections of the script featuring the new API are indicated below.

```
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 avg. thermal excitations
T = pi/(4*g)        # gate period

# construct Hamiltonian
H = g * (tensor(sigmax(), sigmax()) +
         tensor(sigmay(), sigmay()))
# construct inital state
psi0 = tensor(basis(2,1), basis(2,0))

# construct collapse operators
c_ops = []
## qubit 1 collapse operators
sm1 = tensor(sigmam(), qeye(2))
sz1 = tensor(sigmaz(), qeye(2))
c_ops.append(sqrt(g1 * (1+n_th)) * sm1)
c_ops.append(sqrt(g1 * n_th) * sm1.dag())
c_ops.append(sqrt(g2) * sz1)
## qubit 2 collapse operators
sm2 = tensor(qeye(2), sigmam())
sz2 = tensor(qeye(2), sigmaz())
c_ops.append(sqrt(g1 * (1+n_th)) * sm2)
c_ops.append(sqrt(g1 * n_th) * sm2.dag())
c_ops.append(sqrt(g2) * sz2)

# evolve the dissipative system
tlist = linspace(0, T, 100)
medata  = mesolve(H, psi0, tlist, c_ops, [])

## NEW API CALL ##
# extract density matrices from Odedata object
rho_list = medata.states

# get final density matrix for fidelity comparison
rho_final = rho_list[-1]
# calculate expectation values
n1 = expect(sm1.dag() * sm1, rho_list)
n2 = expect(sm2.dag() * sm2, rho_list)
# calculate the ideal evolution
medata_ideal = mesolve(H, psi0, tlist, [], [])

## NEW API CALL ##
# extract states from Odedata object
psi_list = medata_ideal.states

# calculate expectation values
n1_ideal = expect(sm1.dag() * sm1, psi_list)
n2_ideal = expect(sm2.dag() * sm2, psi_list)
# get last ket vector for comparison
psi_ideal = psi_list[-1]
# output is ket since no collapse operators.
rho_ideal = ket2dm(psi_ideal)

# calculate the fidelity of final states
F = fidelity(rho_ideal, rho_final)
```

4.2. Floquet modes of a driven two-level system

Following the example in Ref. [7], here we calculate the quasi-energies for the time-dependent Floquet basis vectors of a sinusoidally driven two-level system [8] with Hamiltonian

$$H = \frac{\Delta}{2}\sigma_z + \frac{E}{2}\cos(\omega t)\,\sigma_x, \tag{1}$$

where Δ is the qubit energy splitting and ω is the driving frequency, for different values of the driving amplitude E. The results of the simulation are presented in Fig. 1.

```
from qutip import *

delta = 1.0 * 2 * pi  # bare qubit sigma_z coefficient
omega = 8.0 * 2 * pi  # driving frequency
T     = (2*pi)/omega  # driving period

# vector of driving amplitudes
E_vec = linspace(0.0, 12.0, 100) * omega

# generate spin operators
sx = sigmax()
sz = sigmaz()

# create array for storing energy values
q_energies = zeros((len(E_vec), 2))

# define time-independent Hamiltonian term
H0 = delta/2.0 * sz
args = {'w': omega}

# loop over driving amplitudes
for idx, E in enumerate(E_vec):
    # amplitude-dependent Hamiltonian term
```

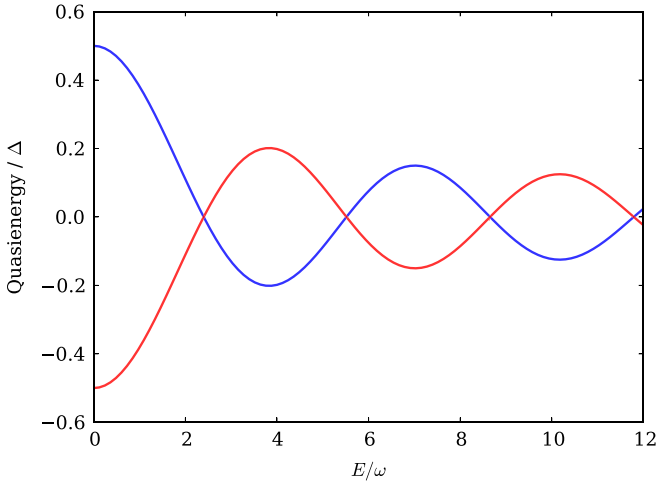

Fig. 1. (Color) Quasienergies corresponding to the two Floquet basis states of a driven two-level system as the driving strength is increased. Here, the quasienergy and driving amplitude are expressed in units of the qubit energy splitting and driving frequency ($\hbar = 1$), respectively. In this simulation, $\Delta = 1 \times 2\pi$ and $\omega = 8 \times 2\pi$.

```
H1 = E/2.0 * sx

# H = H0 + H1 * cos(w * t) in 'list-string' format
H = [H0, [H1, 'cos(w * t)']]

# find the Floquet modes
f_modes, f_energies = floquet_modes(H, T, args)
# record quasi-energies
q_energies[idx,:] = f_energies
```

4.3. Floquet evolution

A driven system that is interacting with its environment is not necessarily well described by the standard Lindblad master equation as its dissipation process could be time dependent due to the driving. In such cases, a rigorous approach would be to take the driving into account when deriving the master equation. This can be done in many different ways, but one common approach is to derive the master equation in the Floquet basis, the Floquet–Markov master equation [9]. In QuTiP, this Floquet–Markov master equation is implemented in the `fmmesolve` function. As this approach is for time-dependent systems, here we model a sinusoidally driven qubit with Hamiltonian

$$H = -\frac{\Delta}{2}\sigma_x - \frac{\epsilon}{2}\sigma_z - A\sigma_x \sin\omega t, \tag{2}$$

where Δ and ϵ are the coupling and energy splitting constants, while A and ω are the driving strength and frequency, respectively. In addition, we define the spectral density of the environmental noise to be Ohmic. In Fig. 2, we plot the occupation probability of the qubit for both the Lindblad and Floquet–Markov master equations as a function of time.

```
from qutip import *

gamma1 = 0.05         # relaxation rate
gamma2 = 0.0          # dephasing  rate
delta = 0.0 * 2 * pi  # qubit sigma_x coefficient
eps0  = 1.0 * 2 * pi  # qubit sigma_z coefficient
A     = 0.1 * 2 * pi  # driving amplitude
w     = 1.0 * 2 * pi  # driving frequency
T     = 2*pi / w      # driving period
psi0  = basis(2,0)    # initial state
tlist = linspace(0, 25.0, 250)

def J_cb(omega):
```

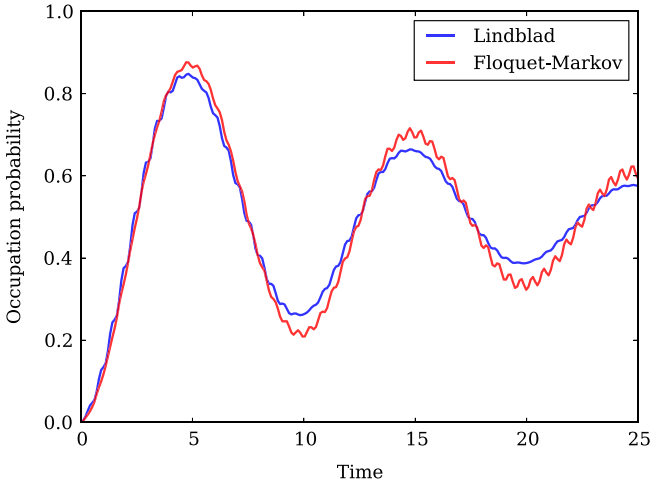

Fig. 2. (Color) Occupation probability of a sinusoidally driven qubit, initially in its ground state, under both Lindblad and Floquet–Markov master equation evolution, where the qubit parameters are $\Delta = 0, \epsilon = 1.0 \times 2\pi$, and the relaxation and dephasing rates are given by $\gamma_1 = 0.05, \gamma_2 = 0$. The driving term has amplitude $A = 0.1 \times 2\pi$ and frequency $\omega = 1.0 \times 2\pi$. Here, the spectral noise density of the environment is assumed to be Ohmic.

```
    """ Noise spectral density """
    return 0.5 * gamma1 * omega/(2*pi)

# Hamiltonian in list-string format
args = {'w': w}
H0 = - delta/2.0 * sigmax() - eps0/2.0 * sigmaz()
H1 = - A * sigmax()
H = [H0, [H1, 'sin(w * t)']]

# ------------------------------------------------------
# Lindblad equation with time-dependent Hamiltonian
#
c_ops = [sqrt(gamma1) * sigmax(),
         sqrt(gamma2) * sigmaz()]
p_ex_me = mesolve(H, psi0, tlist, c_ops,
                  [num(2)], args=args).expect[0]

# ------------------------------------------------------
# Floquet-Markov master equation dynamics
#
rhs_clear() # clears previous time-dependent Hamiltonian

# find initial Floquet modes and quasienergies
f_modes_0, f_energies = floquet_modes(H, T, args, False)

# precalculate Floquet modes for the first driving period
f_modes_table = floquet_modes_table(f_modes_0, f_energies,
                    linspace(0, T, 500+1), H, T, args)

# solve the Floquet-Markov master equation
rho_list = fmmesolve(H, psi0, tlist, [sigmax()]
                     [], [J_cb], T, args).states

# calculate expectation values in the computational basis
p_ex_fmme = zeros(shape(p_ex_me))
for idx, t in enumerate(tlist):
    f_modes_t = floquet_modes_t_lookup(f_modes_table, t, T)
    p_ex_fmme[idx] = expect(num(2),
                rho_list[idx].transform(f_modes_t, False))
```

4.4. Bloch–Redfield master equation

The Lindblad master equation is constructed so that it describes a physical evolution of the density matrix (i.e., trace and positivity preserving), but it does not provide a connection to any underlying microscopic physical model. However, a microscopic model can in some cases be advantageous, as for example in systems with varying energy biases and eigenstates that couple to an environment in some well-defined manner, through a physically

motivated system–environment interaction operator that can be related to a noise–power spectrum. The Bloch–Redfield formalism is one such approach to derive a master equation from the underlying microscopic physics of the system–bath coupling. To highlight the differences inherent in these two approaches, in Fig. 3 we plot the expectation values for the spin operators of the first qubit in a coupled qubit system given by the Hamiltonian

$$H = \omega_1 \left[\cos\theta_1 \sigma_z^{(1)} + \sin\theta_1 \sigma_x^{(1)}\right] + \omega_2 \left[\cos\theta_2 \sigma_z^{(2)} + \sin\theta_2 \sigma_x^{(2)}\right] + g\sigma_x^{(1)}\sigma_x^{(2)} \quad (3)$$

with the initial state $|\phi\rangle = |\psi_1\rangle\,|\psi_2\rangle$ with

$$|\psi_1\rangle = \frac{5}{\sqrt{17}}\left[0.8|0\rangle_{(1)} + (1-0.8)|1\rangle_{(1)}\right]$$

$$|\psi_2\rangle = \frac{5}{\sqrt{17}}\left[(1-0.8)|0\rangle_{(2)} + 0.8|1\rangle_{(2)}\right],$$

where the subscripts indicate which qubit the state belongs to. In Eq. (3), g is the qubit coupling, ω_1 and ω_2 are the qubit frequencies, and finally θ_1 and θ_2 represent the angles of each qubit with respect to the σ_z direction. In this example, the qubit environments in the Bloch–Redfield simulation are assumed to have an Ohmic spectrum. The code for the corresponding simulation is given below.

```
from qutip import *

w     = array([1.0,1.0])*2*pi    # qubit angular frequency
theta = array([0.025,0.0])*2*pi  # angle from sigma_z axis
gamma1 = [0.25, 0.35]            # qubit relaxation rate
gamma2 = [0.0, 0.0]              # qubit dephasing rate
g      = 0.1 * 2 * pi            # coupling strength
# initial state
a = 0.8
psi1 = (a*basis(2,0)+(1-a)*basis(2,1)).unit()
psi2 = ((1-a)*basis(2,0)+a*basis(2,1)).unit()
psi0 = tensor(psi1, psi2)
# times at which to evaluate expectation values
tlist = linspace(0, 15, 500)

# operators for qubit 1
sx1 = tensor(sigmax(), qeye(2))
sy1 = tensor(sigmay(), qeye(2))
sz1 = tensor(sigmaz(), qeye(2))
sm1 = tensor(sigmam(), qeye(2))
# operators for qubit 2
sx2 = tensor(qeye(2), sigmax())
sy2 = tensor(qeye(2), sigmay())
sz2 = tensor(qeye(2), sigmaz())
sm2 = tensor(qeye(2), sigmam())
# Hamiltonian
# qubit 1
H  = w[0] * (cos(theta[0]) * sz1 + sin(theta[0]) * sx1)
# qubit 2
H += w[1] * (cos(theta[1]) * sz2 + sin(theta[1]) * sx2)
# interaction term
H += g * sx1 * sx2

# ------------------------------------------------------
# Lindblad master equation
#
c_ops = []
c_ops.append(sqrt(gamma1[0]) * sm1)
c_ops.append(sqrt(gamma1[1]) * sm2)

lme_results = mesolve(H, psi0, tlist, c_ops,
                      [sx1, sy1, sz1])

# ------------------------------------------------------
# Bloch-Redfield master equation
#
def ohmic_spectrum1(w):
    if w == 0.0:
        # dephasing inducing noise
        return 0.5 * gamma2[0]
    else:
        # relaxation inducing noise
        return 0.5 * gamma1[0]*w/(2*pi)*(w > 0.0)

def ohmic_spectrum2(w):
    if w == 0.0:
        # dephasing inducing noise
        return 0.5 * gamma2[1]
    else:
        # relaxation inducing noise
        return 0.5 * gamma1[1]*w/(2*pi)*(w > 0.0)

brme_results = brmesolve(H, psi0, tlist, [sx1, sx2],
                  [sx1, sy1, sz1], [ohmic_spectrum1,
                  ohmic_spectrum2])
```

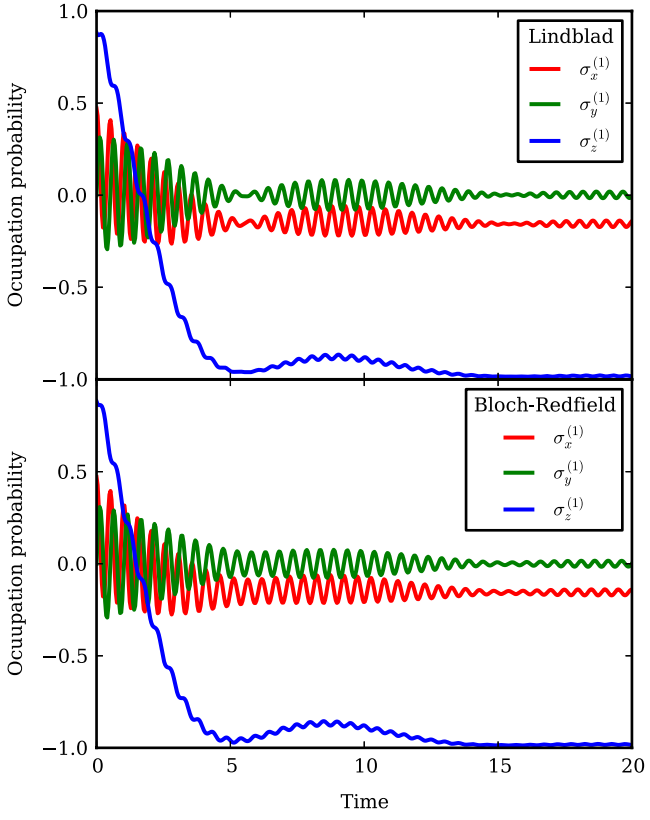

Fig. 3. (Color) Expectation values for the spin operators of qubit 1, for both the Lindblad master equation (top) and the Bloch–Redfield master equation (bottom), where the qubit environment in the latter case is assumed to have an Ohmic spectrum. Here, $\omega_1 = \omega_2 = 1.0 \times 2\pi$, $\theta_1 = 0.025 \times 2\pi$, $\theta_2 = 0$, and the qubit relaxation terms for qubit 1 and qubit 2 are give by $\gamma_1^{(1)} = 0.25$ and $\gamma_1^{(2)} = 0.35$, respectively. The qubit coupling is $g = 0.05 \times 2\pi$. In this simulation, the dephasing terms are assumed to be zero.

4.5. Quantum process tomography

To demonstrate quantum process tomography, here we simulate the effects of relaxation and dephasing on the two-qubit iSWAP gate [10] when the qubits are coupled to a thermal environment with on average $\langle n\rangle = 1.5$ excitations. The χ-matrix obtained from QPT contains all the information about the dynamics of this open quantum system. In Fig. 4, we plot the χ-matrix for both the dissipative and corresponding ideal (unitary) iSWAP gate dynamics.

```
from qutip import *

g = 1.0 * 2 * pi  # coupling strength
g1 = 0.75         # relaxation rate
g2 = 0.25         # dephasing rate
```

```
n_th = 1.5        # bath avg. thermal excitations
T = pi/(4*g)      # gate period

H = g*(tensor(sigmax(),sigmax())+tensor(sigmay(),sigmay()))
psi0 = tensor(basis(2,1), basis(2,0))

c_ops = []
# qubit 1 collapse operators
sm1 = tensor(sigmam(), qeye(2))
sz1 = tensor(sigmaz(), qeye(2))
c_ops.append(sqrt(g1 * (1+n_th)) * sm1)
c_ops.append(sqrt(g1 * n_th) * sm1.dag())
c_ops.append(sqrt(g2) * sz1)
# qubit 2 collapse operators
sm2 = tensor(qeye(2), sigmam())
sz2 = tensor(qeye(2), sigmaz())
c_ops.append(sqrt(g1 * (1+n_th)) * sm2)
c_ops.append(sqrt(g1 * n_th) * sm2.dag())
c_ops.append(sqrt(g2) * sz2)

# process tomography basis
op_basis = [[qeye(2), sigmax(), sigmay(), sigmaz()]] * 2
op_label = [["i", "x", "y", "z"]] * 2

# dissipative gate
U_diss = propagator(H, T, c_ops)
chi = qpt(U_diss, op_basis)
qpt_plot_combined(chi, op_label)

# ideal gate
U_psi = (-1j * H * T).expm()
U_ideal = spre(U_psi) * spost(U_psi.dag())
chi = qpt(U_ideal, op_basis)
qpt_plot_combined(chi, op_label)
```

4.6. Exporting QuTiP data

Finally, we demonstrate the exporting of data generated in QuTiP to an external plotting program using `file_data_store` and `file_data_read` to save and load the data, respectively. To keep the example completely in Python, we have chosen to use Mayavi [11] to plot a Wigner function in Fig. 5 generated in QuTiP corresponding to the state $|\Psi\rangle = |\alpha\rangle + |-\alpha\rangle + |\tilde{\phi}\rangle$ that is a Schrödinger cat state consisting of two coherent states with complex amplitude α, with an additional random ket vector $|\tilde{\phi}\rangle$ created using QuTiP's random state generator:

```
from qutip import *
# Number of basis states
N = 20
# amplitude of coherent states
alpha = 2.0 + 2.0j
# define ladder operators
a = destroy(N)
# define displacement operators
D1 = displace(N, alpha)
D2 = displace(N, -alpha)
# create superposition of coherent states + random ket
psi = (D1 + D2) * basis(N,0) + 0.5 * rand_ket(N)
psi = psi.unit() # normalize
# calculate Wigner function
xvec = linspace(-5, 5, 500)
yvec = xvec
W = wigner(psi, xvec, yvec)

## new function calls ##
# store Wigner function to file
file_data_store("wigner.dat", W, numtype='real')
# load input data from file
input_data = file_data_read('wigner.dat')

# plot using mayavi
from mayavi.mlab import *
X,Y = meshgrid(xvec, yvec)
surf(xvec, yvec, input_data, warp_scale='auto')
view(0, 45)
show()
```

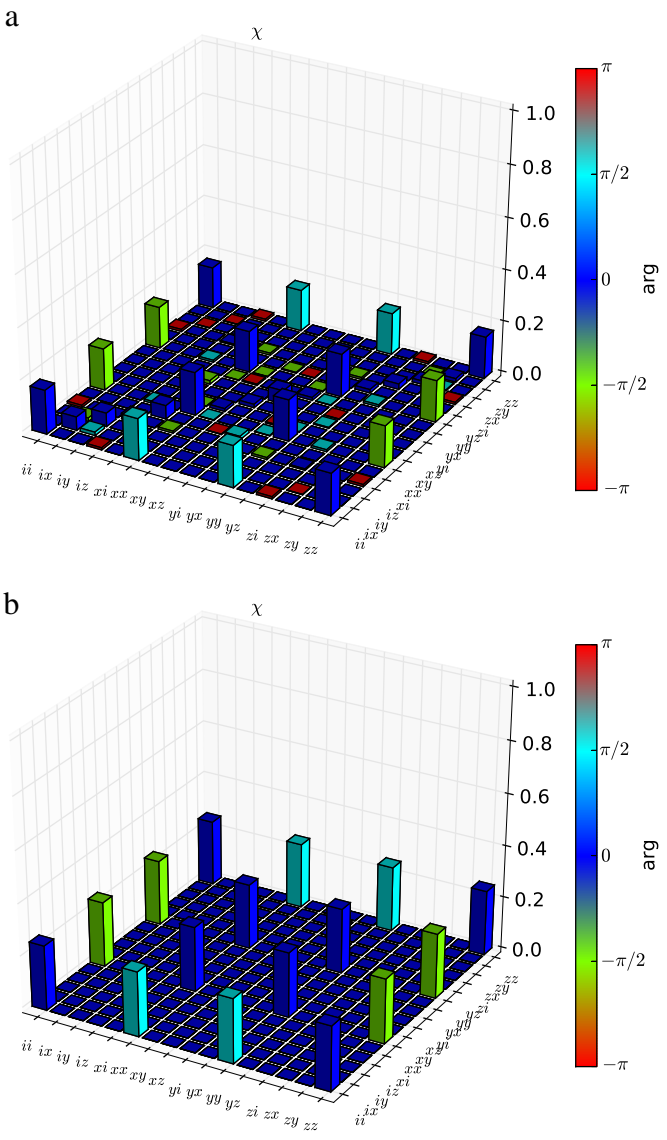

Fig. 4. (Color) (a) The QPT process χ-matrix for the dissipative iSWAP gate between two qubits. Here, the color indicates the phase of each matrix element. The qubit–qubit coupling strength is $g = 1.0 \times 2\pi$, whereas the relaxation and dephasing rates are $g1 = 0.75$ and $g2 = 0.25$, respectively. (b) The ideal iSWAP gate χ-matrix when the qubit dissipation and dephasing are not present.

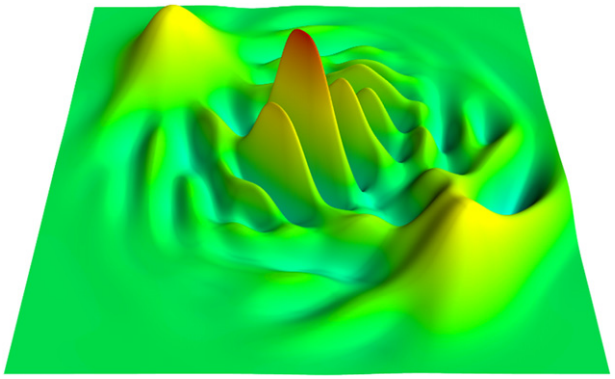

Fig. 5. (Color) Wigner function for the state $|\Psi\rangle = |\alpha\rangle + |-\alpha\rangle + |\tilde{\phi}\rangle$, where $\alpha = 2 + 2j$ is the coherent state amplitude, and $|\tilde{\phi}\rangle$ is a randomly generated state vector. This plot is generated in Mayavi using data from the QuTiP `wigner` function.

Table A.1
List of new user-accessible functions available in QuTiP 2.1. Additional information about each function may be obtained by calling `help(function_name)` from the Python command line, or at the QuTiP website [2].

Quantum object methods	
`conj`	Conjugate of quantum object.
`eigenenergies`	Calculates the eigenvalues (eigenenergies if the operator is a Hamiltonian) for a quantum operator.
`groundstate`	Returns the eigenvalue and eigenstate corresponding to the ground state of the quantum operator.
`matrix_element`	Gives the matrix element $Q_{nm} = \langle\psi_n\|\hat{Q}\|\psi_m\rangle$ for the given operator and states ψ_n and ψ_m.
`tidyup`	Removes the small elements from a quantum object.
`trans`	Transpose of a quantum object.
Bloch–Redfield functions	
`bloch_redfield_solve`	Evolve the ODEs defined by the Bloch–Redfield tensor.
`bloch_redfield_tensor`	Bloch–Redfield tensor for a set of system-bath operators and corresponding spectral functions.
Floquet functions	
`floquet_modes`	Calculate the initial Floquet modes given a periodic time-dependent Hamiltonian.
`floquet_modes_t`	Calculate the Floquet modes at a time t.
`floquet_modes_table`	Calculate a table of Floquet modes for an interval of times.
`floquet_modes_t_lookup`	Look up the Floquet modes at an arbitrary time t given a pre-computed Floquet-mode table.
`floquet_states`	Calculate the initial Floquet states given a set of Floquet modes.
`floquet_states_t`	Calculate the Floquet states for an arbitrary time t given a set of Floquet modes.
`floquet_state_decomposition`	Decompose an arbitrary state vector in the basis of the given Floquet modes.
`floquet_wavefunction`	Calculate the initial wavefunction given a Floquet-state decomposition and Floquet modes.
`floquet_wavefunction_t`	Calculate the wavefunction for an arbitrary time t given a Floquet-state decomposition and modes.
Evolution solvers	
`brmesolve`	Bloch–Redfield master equation solver.
`fmmesolve`	Floquet–Markov master equation solver.
Correlation functions	
`correlation`	Transient two-time correlation function.
`correlation_ss`	Steady-state two-time correlation function.
Entropy/entanglement functions	
`concurrence`	Calculate the concurrence entanglement measure for a two-qubit state.
`entropy_conditional`	The conditional entropy $S(A\|B) = S(A, B) - S(B)$ of a selected density matrix component.
`entropy_mutual`	Mutual information $S(A : B)$ between selection components of a system density matrix.
Quantum process tomography	
`qpt`	Quantum process tomography χ-matrix for a given (possibly non-unitary) transformation matrix U.
`qpt_plot`	Visualize the quantum process tomography χ-matrix. Plot the real and imaginary parts separately.
`qpt_plot_combined`	χ-matrix plot with height and color corresponding to the absolute value and phase, respectively.
Random state/operator generation	
`rand_dm`	Random $N \times N$ density matrix.
`rand_herm`	Random $N \times N$ Hermitian operator.
`rand_ket`	Random $N \times 1$ state (ket) vector.
`rand_unitary`	Random $N \times N$ Unitary operator.
Gates	
`iswap`	Quantum object representing the *i*SWAP gate.
`sqrtiswap`	Quantum object representing the square root *i*SWAP gate.
Utility functions	
`file_data_read`	Retrieves an array of data from the requested file
`file_data_store`	Stores a matrix of data to a file to be read by an external program.
`qload`	Loads quantum object or array data contained from given file.
`qsave`	Saves any quantum object or array data to the specified file.
`rhs_generate`	Pre-compiles the Cython code for time-dependent `mesolve` problems run inside a `parfor` loop.
`rhs_clear`	Clears string-format time-dependent Hamiltonian data.

Acknowledgments

JRJ and PDN were supported by the Japanese Society for the Promotion of Science (JSPS) Foreign Postdoctoral Fellowship No. P11501 and P11202, respectively. PDN also acknowledges support from Kakenhi grant number 2301202 and Korea University. FN acknowledges partial support from the ARO, National Science Foundation (NSF) grant No. 0726909, Grant-in-Aid for Scientific Research (S), MEXT Kakenhi on Quantum Cybernetics, and the JSPS-FIRST program.

Appendix. New functions in QuTiP 2.1

See Table A.1.

References

[1] J.R. Johansson, P.D. Nation, F. Nori, Comput. Phys. Commun. 183 (2012) 1760.
[2] P.D. Nation, J.R. Johansson, QuTiP: Quantum Toolbox in Python, 2011–2012. http://code.google.com/p/qutip/.
[3] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn, K. Smith, Computing in Science & Engineering 13 (2011) 31.
[4] J.R. Johansson, P.D. Nation, F. Nori, 2013, unpublished.
[5] M. Mohseni, A.T. Rezakhani, D.A. Lidar, Phys. Rev. A 77 (2008) 032322.
[6] J.D. Hunter, Computing in Science & Engineering 9 (2007) 90.
[7] C.E. Creffield, Phys. Rev. B 67 (2003) 165301.
[8] S.N. Shevchenko, S. Ashhab, F. Nori, Phy. Rep. 492 (2010) 1.
[9] M. Grifoni, P. Hänggi, Phys. Rep. 304 (1998) 229.
[10] N. Schuch, J. Siewert, Phys. Rev. A 67 (2003) 032301.
[11] P. Ramachandran, G. Varoquaux, Computing in Science & Engineering 13 (2011) 40.