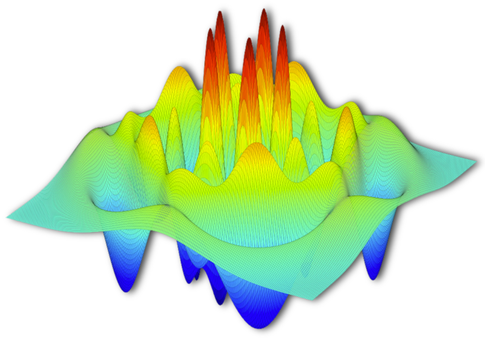


QuTiP v2.2

The Quantum Toolbox in Python

J.R. Johansson (RIKEN), P.D. Nation (Korea University), F. Nori (RIKEN, U. Michigan)

http://www.qutip.org

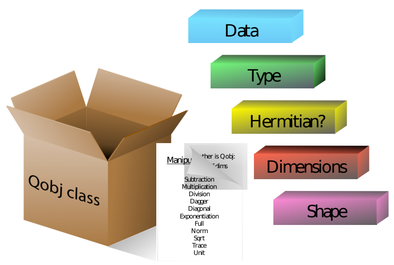
About QuTiP

QuTiP is an open-source software framework for simulating the dynamics of closed and open quantum systems. The QuTiP library depends on the excellent Numpy and Scipy numerical packages. In addition, graphical output is provided by Matplotlib. QuTiP aims to provide user-friendly and efficient numerical simulations of a wide variety of quantum mechanical problems, including those with Hamiltonians and/or collapse operators with arbitrary time-dependence, commonly found in a wide range of physics applications, such as quantum optics, trapped ions, superconducting circuits, semiconducting quantum dots, and quantum nanomechanical resonators. QuTiP is freely available for use and/or modification on all Unix-based platforms and on Windows.

i - Quantum Objects

A core feature in QuTiP, the Qobj class, provides an representation of quantum objects, such as operators, density matrices, state vectors (bras and kets). In addition, a large number of operations that one commonly need to perform on quantum objects are implemented as methods in the Qobj class. For example:

- artihmetic operations
- conjugate and transpose
- groundstate, eigenstates and energies
- basis transformations
- matrix elements
- partial trace and partial transpose
- extract and eliminate states
- matrix exponentiation
- norm and normalization



Quantum objects

```
>>> psi = Qobj([1, 0])  # create a bra state vector
  Quantum object: dims = [[1], [2]], shape = [1, 2], type = ket
  Qobj data =
  [[ 1.  0.]]

>>> psi.dag()
  Quantum object: dims = [[2], [1]], shape = [2, 1], type = ket
  Qobj data =
  [[ 1.]
   [ 0.]]
```

Composite systems

```
>>> H = tensor(sigmax(), sigmax())
  Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isherm = True
  Qobj data =
  [[ 0.  0.  0.  1.]
   [ 0.  0.  1.  0.]
   [ 0.  1.  0.  0.]
   [ 1.  0.  0.  0.]]

>>> psi = tensor([fock(2, 1), fock(2, 0)])
>>> ptrace(psi, 0)
  Quantum object: dims = [[2], [2]], shape = [2, 2], type = oper, isherm = True
  Qobj data =
  [[ 0.  0.]
   [ 0.  1.]]
```

ii. States, operators, gates

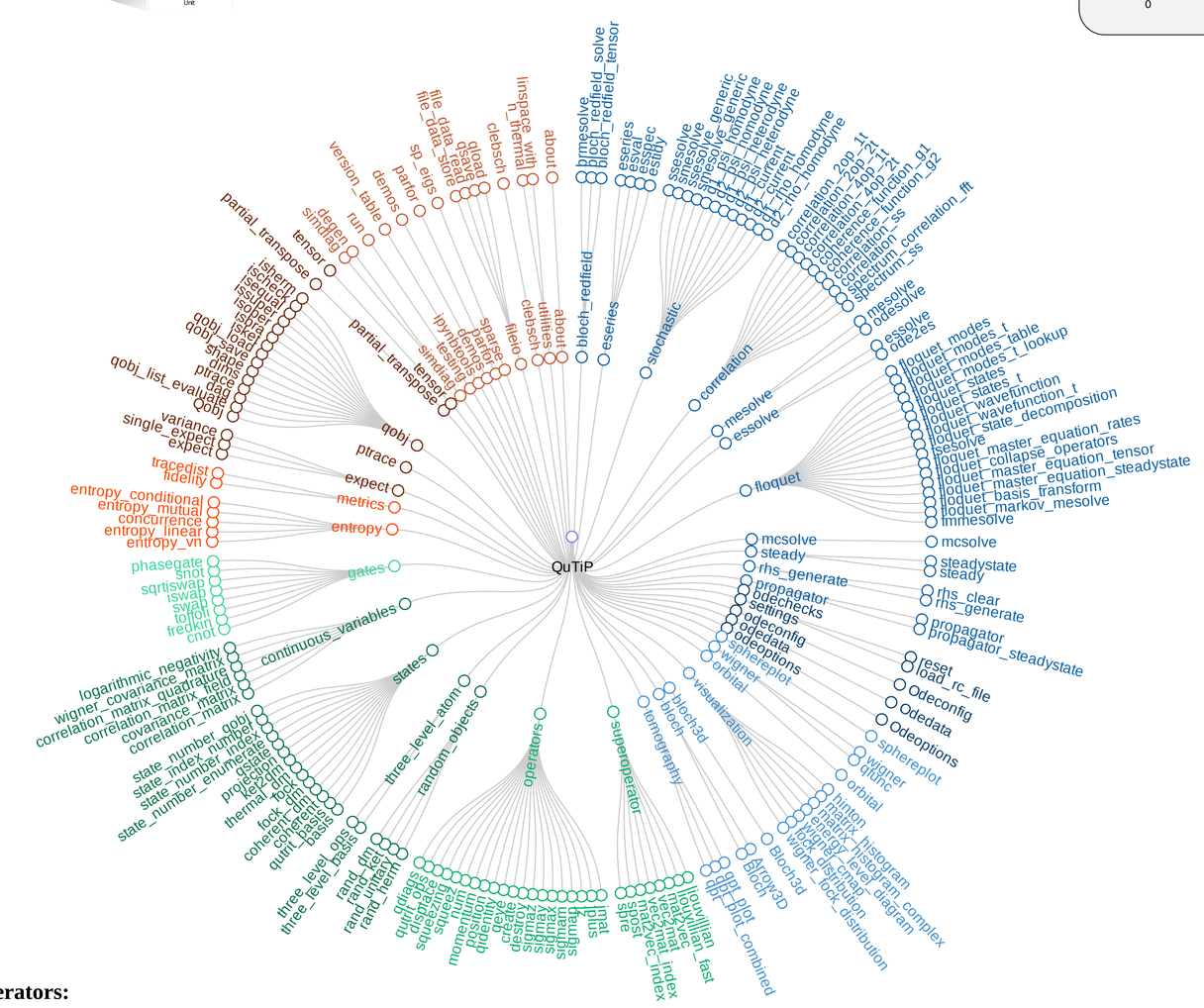
QuTiP includes a large library of functions that creats Qobj representations of common quantum states, operators and superoperators:

States:
- Coherent states: `coherent(N, alpha)`
- Fock states: `fock(N, n)`
- Thermal states: `thermal_dm(N, n_th)`

Operators:
- Idenity: `identity(N)`
- Annihilation: `destroy(N)`
- Creation: `create(N)`
- Number: `num(N)`
- Displacement: `displace(N, alpha)`
- Pauli operators: `sigmax(), sigmay() sigmaz(), sigmam(), sigmap()`
- Spin operators: `jmat(k, s)`

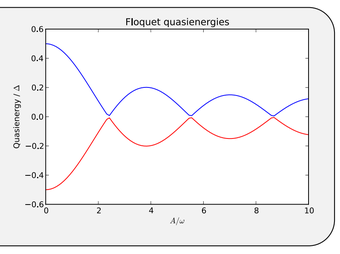
Superoperators:
- Liouvillian: `liouvillian(H, c_ops)`
- Pre- and post: `spre(op), spost(op)`

Gates:
- Not: `cnot(), snot()`
- Swap: `iswap(), swap(), sqrtiswap()`
- Phase: `phasegate(theta)`
- 3-qubit gates: `fredkin(), toffoli()`



Floquet energies

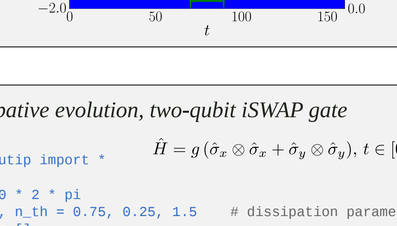
```
from qutip import *

H0 = delta/2.0 * sigmaz() - eps0/2.0 * sigmax()
H1 = A/2.0 * sigmax()
for idx, A in enumerate(A_vec):
   H = [H0, [H1, lambda t, w: sin(w*t)]]
   f_modes, f_energies = \
       floquet_modes(H, T, args=omega, True)
   q_energies[idx,:] = f_energies
```



Driven qubit, time-dependent Hamiltonian

$$\hat{H}(t) = -\frac{\Delta}{2}\hat{\sigma}_z - \frac{\epsilon}{2}\hat{\sigma}_x - A\cos(\omega t)\hat{\sigma}_z$$

```
from qutip import *

epsilon = delta = 1.0
psi0 = basis(2,0)                 # Initial state: ground state
H0 = - delta * sigmaz() - epsilon * sigmax()
H1 = - sigmaz()
h_t = [H0, [H1, 'A * cos(w*t)']]       # Hamiltonian
args = {'A': 10.017, 'w': 0.025*2*pi}
e_ops = [sigmax(), sigmay(), sigmaz()] # Expectation values

output = sesolve(h_t, psi0, linspace(0, 160, 500), e_ops, args)
```



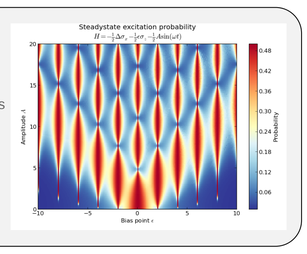
Dissipative evolution, two-qubit iSWAP gate

$$\hat{H} = g\left(\hat{\sigma}_x \otimes \hat{\sigma}_x + \hat{\sigma}_y \otimes \hat{\sigma}_y\right),\ t \in [0, T = \pi/4g]$$

```
from qutip import *

g = 1.0 * 2 * pi
g1, g2, n_th = 0.75, 0.25, 1.5    # dissipation parameters
c_ops = []
H = g * (tensor(sigmax(), sigmax()) + tensor(sigmay(), sigmay()))

sm1 = tensor(sigmam(), qeye(2))   # qubit 1 collapse operators
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)

sm2 = tensor(qeye(2), sigmam())   # qubit 2 collapse operators
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)

U = propagator(H, pi/(4*g), c_ops)
```

Steady states

```
H = ...      # Hamiltonian
c_ops = ... # collapse operators
rho_ss = steadystate(H, c_ops)
```

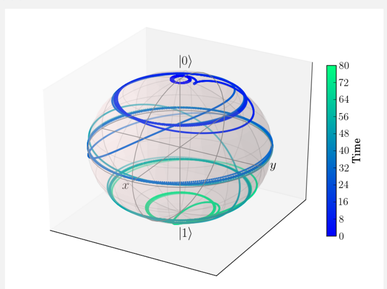


iv. Visualization

Visualization of quantum states and processes is an important part of QuTiP, which includes a large number of classes and functions for visualization techniques commonly used in quantum mechanics. For example:

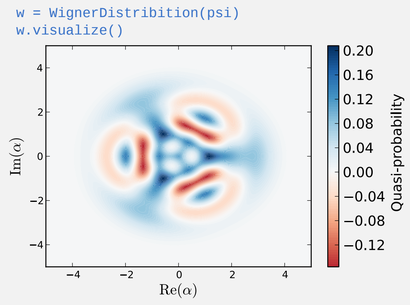
Bloch sphere

```
from qutip import *

b = Bloch()
b.add_points([x_vec, y_vec, z_vec])
...
```



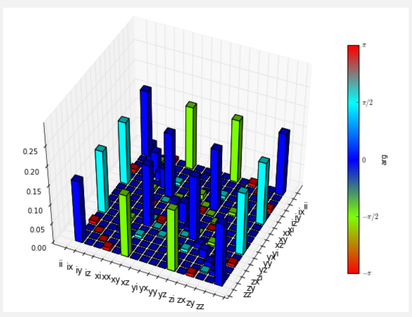
Wigner functions

```
from qutip import *

psi = sum([basis(10, n)
           for n in [0, 3, 6]]).unit()

w = WignerDistribition(psi)
w.visualize()
```



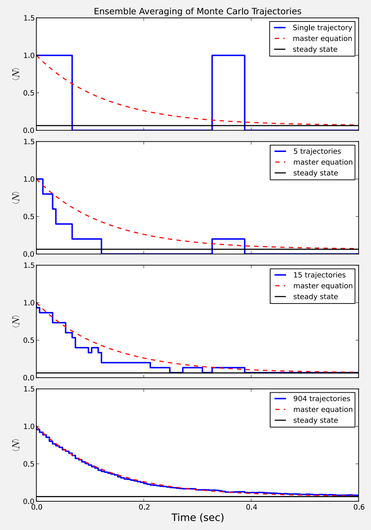
Process tomography

```
from qutip import *

op_basis = [[qeye(2), sigmax(), sigmay(), sigmaz()]
            for i in range(2)]
op_labels = [["i", "x", "y", "z"]
             for i in range(2)]

chi = qpt(U, op_basis)
qpt_plot_combined(chi, op_labels)
```



Stochastic decay of a single Fock state

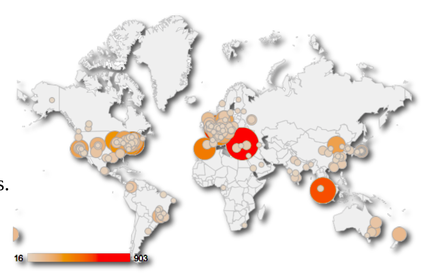
```
from qutip import *

N = 4                # number of basis states
kappa = 1.0/0.129    # coupling to heat bath
nth = 0.063          # temperature with <n>=0.063

a = destroy(N)       # cavity annihilation operator
H = a.dag() * a      # harmonic oscillator Hamiltonian
psi0 = basis(N,1)    # initial state

c_op_list = []
c_op_list.append(sqrt(kappa * (1 + nth)) * a)
c_op_list.append(sqrt(kappa * nth) * a.dag())

ntraj = [1, 5, 15, 904]
tlist = linspace(0,0.6,100)

mc = mcsolve(H, psi0, tlist, c_op_list, [a.dag()*a], ntraj)
me = mesolve(H, psi0, tlist, c_op_list, [a.dag()*a])

# calculate steady state
final_state = steadystate(H, c_op_list)
fexpt = expect(a.dag()*a, final_state)

# plot results ...
```



v. Publications

1. J. R. Johansson, P. D. Nation, and F. Nori,
   QuTiP 2: A Python framework for the dynamics of open quantum systems.
   Comp. Phys. Comm. **184**, 1234 (2013)
2. J. R. Johansson, P. D. Nation, and F. Nori,
   QuTiP: An open-source Python framework for the dynamics of open quantum systems.
   Comp. Phys. Comm. **183**, 1760–1772 (2012)

More information and download at: `http://www.qutip.org`



RIKEN

KOREA UNIVERSITY 고려대학교

日本学術振興会 Japan Society for the Promotion of Science