QuTiP^2

The Quantum Toolbox in Python

Lecture on

QuTiP: Quantum Toolbox in Python

with case studies in

Circuit-QED

Robert Johansson
RIKEN

In collaboration with
Paul Nation
Korea University

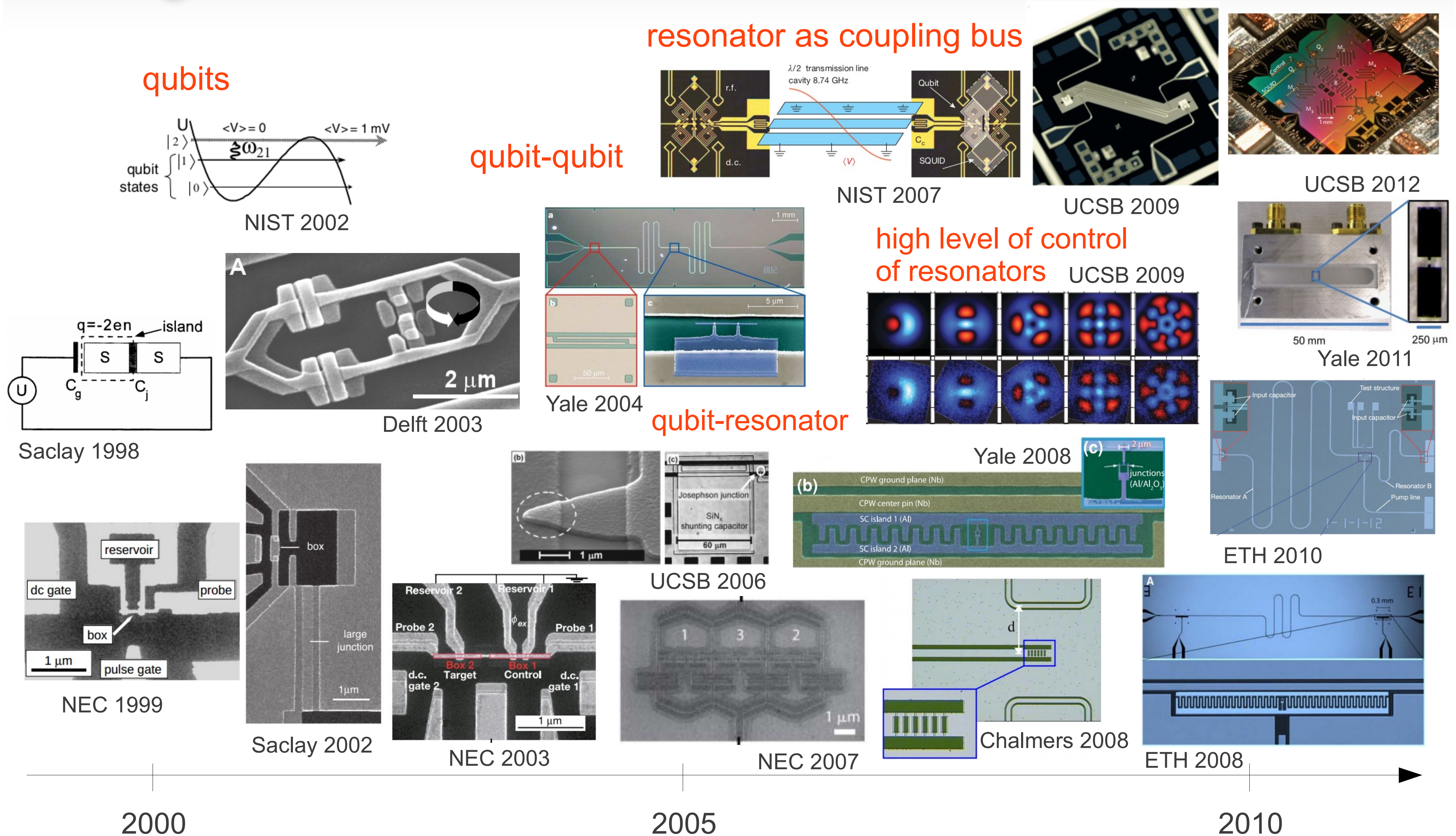
日本学術振興会
Japan Society for the Promotion of Science

Content

- Introduction to QuTiP
- Case studies in circuit-QED
  - Jaynes-Cumming-like models
    - Vacuum Rabi oscillations
    - Qubit-gates using a resonators as a bus
    - Single-atom laser
  - Dicke model / Ultrastrong coupling
  - Correlation functions and nonclassicality tests
  - Parametric amplifier

QuTiP^2

The Quantum Toolbox in Python

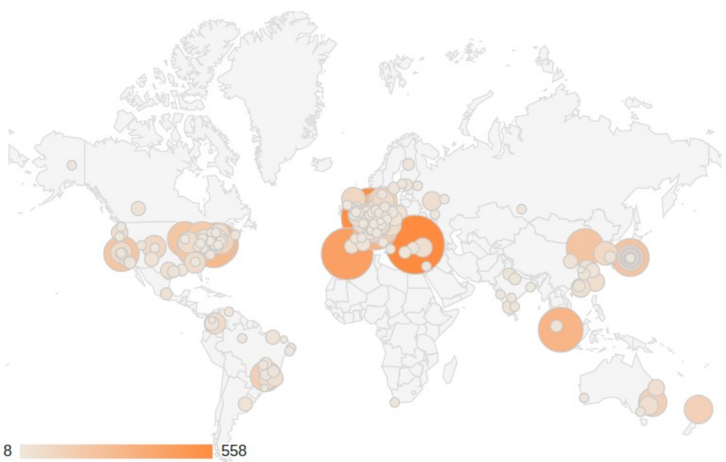


What is QuTiP?

- Framework for computational quantum dynamics
  - Efficient and easy to use for quantum physicists
  - Thoroughly tested (100+ unit tests)
  - Well documented (200+ pages, 50+ examples)
  - Quite large number of users (>1000 downloads)
- Suitable for
  - theoretical modeling and simulations
  - modeling experiments
- 100% open source
- Implemented in Python/Cython using SciPy, Numpy, and matplotlib

Project information

Authors:	Paul Nation and Robert Johansson
Web site:	http://qutip.googlecode.com
Discussion:	Google group “qutip”
Blog:	http://qutip.blogspot.com
Platforms:	Linux and Mac
License:	GPLv3
Download:	http://code.google.com/p/qutip/downloads
Repository:	http://github.com/qutip
Publication:	Comp. Phys. Comm. **183**, 1760 (2012)



What is Python?

Python is a modern, general-purpose, interpreted programming language

Modern

Good support for object-oriented and modular programming, packaging and reuse of code, and other good programming practices.

General purpose

Not only for scientific use. Huge number of top-quality packages for communication, graphics, integration with operating systems and other software packages.

Interpreted

No compilation, automatic memory management and garbage collection, very easy to use and program.

python™

More information:
http://www.python.org

More information at:
http://www.scipy.org

Why use Python for scientific computing?

- Widespread use and a strong position in the computational physics community
- Excellent libraries and add-on packages
  - numpy — for efficient vector, matrix, multidimensional array operations
  - scipy — huge collection of scientific routines
    ode, integration, sparse matrices, special functions, linear algebra, fourier transforms, …
  - matplotlib — for generating high-quality raster and vector graphics in 2D and 3D
- Great performance due to close integration with time-tested and highly optimized compiled codes
  - blas, atlas blas, lapack, arpack, Intel MKL, …
- Modern general purpose programming language with good support for
  - Parallel processing, interprocess communication (MPI, OpenMP), ...

What we want to accomplish with QuTiP

Objectives

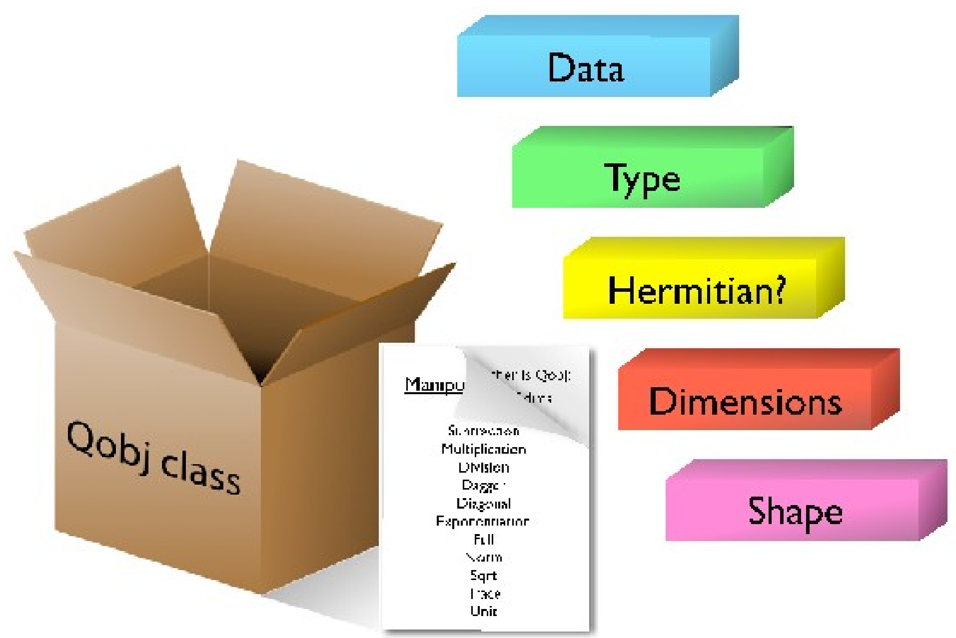
To provide a powerful framework for quantum mechanics that closely resembles the standard mathematical formulation

- Efficient and easy to use
- General framework, able to handle a wide range of different problems

Design and implementation

- Object-oriented design
- Qobj class used to represent quantum objects
  - Operators
  - State vectors
  - Density matrices
- Library of utility functions that operate on Qobj instances

QuTiP core class: Qobj

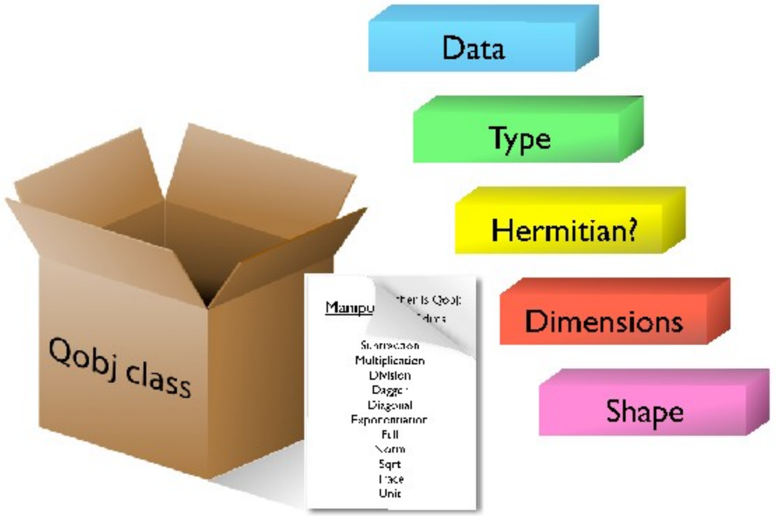


Quantum object class: `Qobj`

Abstract representation of quantum states and operators

- Matrix representation of the object
- Structure of the underlaying state space, Hermiticity, type, etc.
- Methods for performing all common operations on quantum objects:

  `eigs(),dag(),norm(),unit(),expm(),sqrt(),tr(), ...`
- Operator arithmetic with implementations of: +. -, *, ...

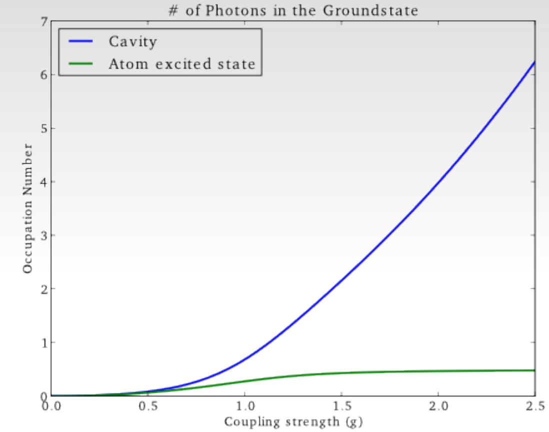


Example: built-in operator $\hat{\sigma}_x$

```
>>> sigmax()

Quantum object: dims = [[2], [2]], shape = [2, 2],
type = oper, isHerm = True
Qobj data =
[[ 0.  1.]
 [ 1.  0.]]
```

Example: built-in state $|\alpha = 0.5\rangle$

```
>>> coherent(5, 0.5)

Quantum object: dims = [[5], [1]], shape = [5, 1], type = ket
Qobj data =
[[ 0.88249693]
 [ 0.44124785]
 [ 0.15601245]
 [ 0.04496584]
 [ 0.01173405]]
```



Calculating using Qobj instances

Basic operations

```
# operator arithmetic
>> H = 2 * sigmaz() + 0.5 * sigmax()

Quantum object: dims = [[2], [2]],
shape = [2, 2], type = oper, isHerm = True
Qobj data =
[[ 2.   0.5]
 [ 0.5 -2. ]]

# superposition states
>> psi = (basis(2,0) + basis(2,1))/sqrt(2)

Quantum object: dims = [[2], [1]],
shape = [2, 1], type = ket
Qobj data =
[[ 0.70710678]
 [ 0.70710678]]


# expectation values
>> expect(num(2), psi)

0.4999999999999999

>> N = 25
>> psi = (coherent(N,1) + coherent(N,3)).unit()
>> expect(num(N), psi)

4.761589143572134
```

Composite systems

```
# operators
>> sx = sigmax()
Quantum object: dims = [[2], [2]],
shape = [2, 2], type = oper, isHerm = True
Qobj data =
[[ 0.  1.]
 [ 1.  0.]]

>> sxsx = tensor([sx,sx])
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.]]

# states
>> psi_a = fock(2,1); psi_b = fock(2,0)
>> psi = tensor([psi_a, psi_b])
Quantum object: dims = [[2, 2], [1, 1]],
shape = [4, 1], type = ket
Qobj data =
[[ 0.]
 [ 1.]
 [ 0.]
 [ 0.]]

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

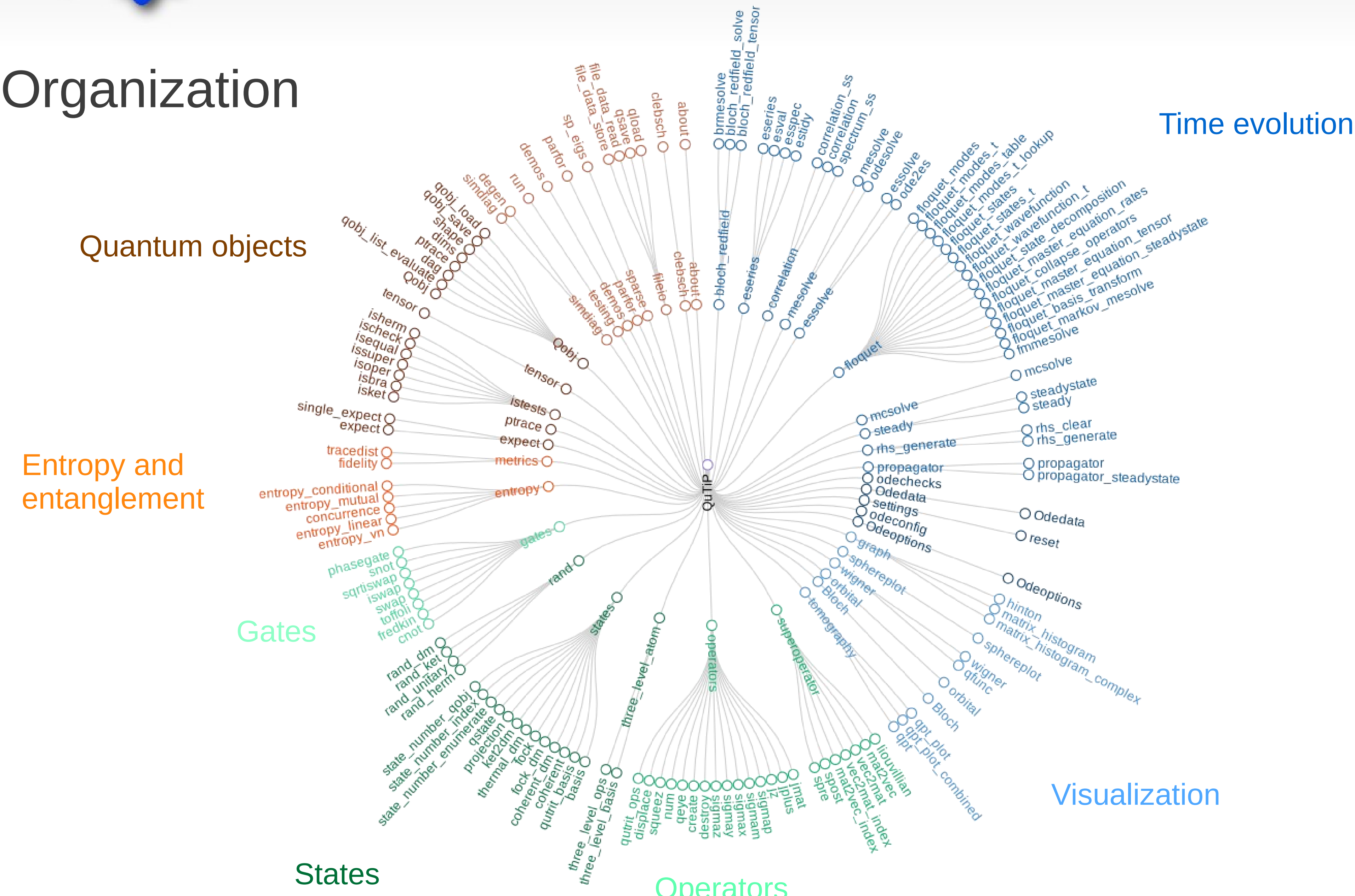
Basis transformations

```
# eigenstates and values for a Hamiltonian
>> H = sigmax()
>> evals, evecs = H.eigenstates()
>> evals

array([-1.,  1.])

>> evecs

array([
Quantum object: dims = [[2], [1]],
shape = [2, 1], type = ket
Qobj data =
[[-0.70710678]
 [ 0.70710678]],
Quantum object: dims = [[2], [1]],
shape = [2, 1], type = ket
Qobj data =
[[ 0.70710678]
 [ 0.70710678]]], dtype=object)

# transform an operator to the eigenbasis of H
>> sx_eb = sigmax().transform(evecs)

Quantum object: dims = [[2], [2]],
shape = [2, 2], type = oper, isHerm = True
Qobj data =
[[-1.  0.]
 [ 0.  1.]]
```

QuTiP²

The Quantum Toolbox in Python

Organization



Evolution of quantum systems

The main use of QuTiP is quantum evolution. A number of solvers are available.

Typical simulation workflow:

i. Define parameters that characterize the system
ii. Create Qobj instances for operators and states
iii. Create Hamiltonian, initial state and collapse operators, if any
iv. Choose a solver and evolve the system
v. Post-process, visualize the data, etc.

Available evolution solvers:

- Unitary evolution: Schrödinger and von Neumann equations
- Lindblad master equations
- Monte-Carlo quantum trajectory method
- Bloch-Redfield master equation
- Floquet-Markov master equation
- Propagators

Lindblad master equation

Equation of motion for the density matrix $\rho(t)$ for a quantum system that interacts with its environment:

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

$$H(t) = \text{system Hamiltonian}$$
$$c_n = \sqrt{\gamma_n}a_n \text{ describes the effect of the environment on the system}$$
$$\gamma_n = \text{rate of the environment-system interaction process}$$

How do we solve this equation numerically?

I. Construct the matrix representation of all operators
II. Evolve the ODEs for the unknown elements in the density matrix
III. For example, calculate expectation values for some selected operators for each $\rho(t)$

Lindblad master equation

Equation of motion for the density matrix $\rho(t)$ for a quantum system that interacts with its environment:

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

$$H(t) = \text{system Hamiltonian}$$

$$c_n = \sqrt{\gamma_n} a_n \text{ describes the effect of the environment on the system}$$

$$\gamma_n = \text{rate of the environment-system interaction process}$$

How do we solve this equation numerically in QuTiP?

```
from qutip import *

psi0 = ...                # initial state
H  = ...                  # system Hamiltonian
c_op_list = [...]         # collapse operators
e_op_list = [...]         # expectation value operators

tlist = linspace(0, 10, 100)
result = mesolve(H, psi0, tlist, c_op_list, e_op_list)
```

Monte-Carlo quantum trajectory method

Equation of motion for a single realization of the state vector $|\psi(t)\rangle$ for a quantum system that interacts with its environment:

$$\frac{d}{dt}|\psi(t)\rangle = -\frac{i}{\hbar}H_{\rm eff}|\psi(t)\rangle \qquad H_{\rm eff}(t) = H(t) - \frac{i\hbar}{2}\sum_n c_n^\dagger c_n$$

$$\delta p = \delta t \sum_n \langle\psi(t)|c_n^\dagger c_n|\psi(t)\rangle = \text{reduction of wavefunction norm}$$

$$|\psi(t+\delta t)\rangle = c_n|\psi(t)\rangle / \langle\psi(t)|c_n^\dagger c_n|\psi(t)\rangle^{1/2} = \text{quantum jump with operator } c_n$$

Comparison to the Lindblad master equation (LME)

I. MC uses state vectors instead of density matrices → huge advantage for large quantum systems
II. MC give only one stochastic realization of the state vector dynamics → need to average over many trajectories to get the ensemble average that can be compared to the density matrix.
III. MC is faster than LME for large system, but LME is faster for small system.

Monte-Carlo quantum trajectory method

Equation of motion for a single realization of the state vector $|\psi(t)\rangle$ for a quantum system that interacts with its environment:

$$\frac{d}{dt}|\psi(t)\rangle = -\frac{i}{\hbar}H_{\rm eff}|\psi(t)\rangle \qquad H_{\rm eff}(t) = H(t) - \frac{i\hbar}{2}\sum_n c_n^\dagger c_n$$

$$\delta p = \delta t \sum_n \langle\psi(t)|c_n^\dagger c_n|\psi(t)\rangle = \text{reduction of wavefunction norm}$$

$$|\psi(t+\delta t)\rangle = c_n|\psi(t)\rangle \,/\, \langle\psi(t)|c_n^\dagger c_n|\psi(t)\rangle^{1/2} = \text{quantum jump with operator } c_n$$

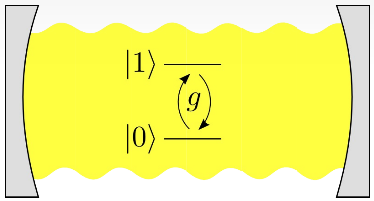
Comparison to the Lindblad master equation (LME) in QuTiP code:

```
from qutip import *

psi0 = ...                # initial state
H  = ...                  # system Hamiltonian
c_list = [...]            # collapse operators
e_list = [...]            # expectation value operators

tlist = linspace(0, 10, 100)
result = mesolve(H, psi0, tlist, c_list, e_list)
```

```
from qutip import *

psi0 = ...                # initial state
H  = ...                  # system Hamiltonian
c_list = [...]            # collapse operators
e_list = [...]            # expectation value operators

tlist = linspace(0, 10, 100)
result = mcsolve(H, psi0, tlist, c_list, e_list, ntraj=500)
```

QuTiP²

The Quantum Toolbox in Python



(a two-level atom in a cavity)

Example: Jaynes-Cummings model

Mathematical formulation:

Hamiltonian

$$\hat{H} = \hbar\omega_c \hat{a}^\dagger \hat{a} - \frac{\hbar\omega_q}{2}\hat{\sigma}_z + \frac{\hbar g}{2}\left(\hat{a}\hat{\sigma}_+ + \hat{a}^\dagger\hat{\sigma}_-\right)$$

Initial state

$$|\psi(t=0)\rangle = |1\rangle_c \otimes |0\rangle_q$$

Time evolution

$$\frac{d}{dt}|\psi(t)\rangle = \hat{H}|\psi(t)\rangle$$

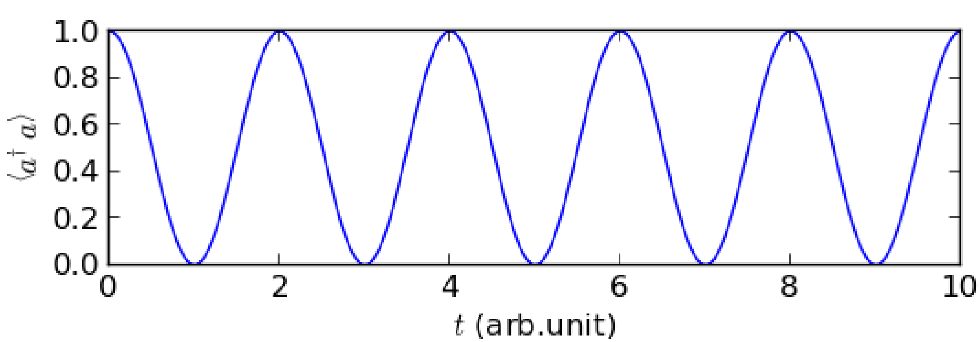
Expectation values

$$\langle \hat{a}^\dagger \hat{a}\rangle = \langle\psi(t)|\, \hat{a}^\dagger \hat{a}\, |\psi(t)\rangle$$

QuTiP code:

```
from qutip import *
N  = 10

a  = tensor(destroy(N),qeye(2))
sz = tensor(qeye(N),sigmaz())
s  = tensor(qeye(N),destroy(2))
wc = wq = 1.0 * 2 * pi
g  = 0.5 * 2 * pi
H  = wc * a.dag() * a - 0.5 * wq * sz + \
     0.5 * g * (a * s.dag() + a.dag() * s)
psi0  = tensor(basis(N,1), basis(2,0))
tlist = linspace(0, 10, 100)
out   = mesolve(H, psi0, tlist, [], [a.dag()*a])

from pylab import *
plot(tlist, out.expect[0])
show()
```

Qobj instances

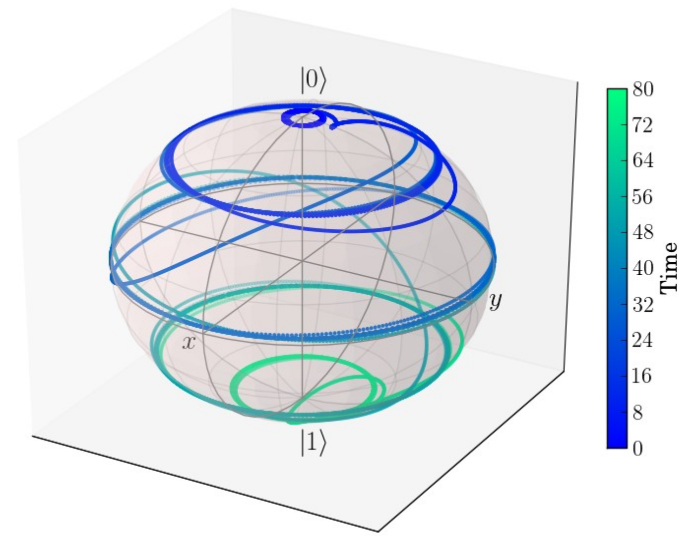


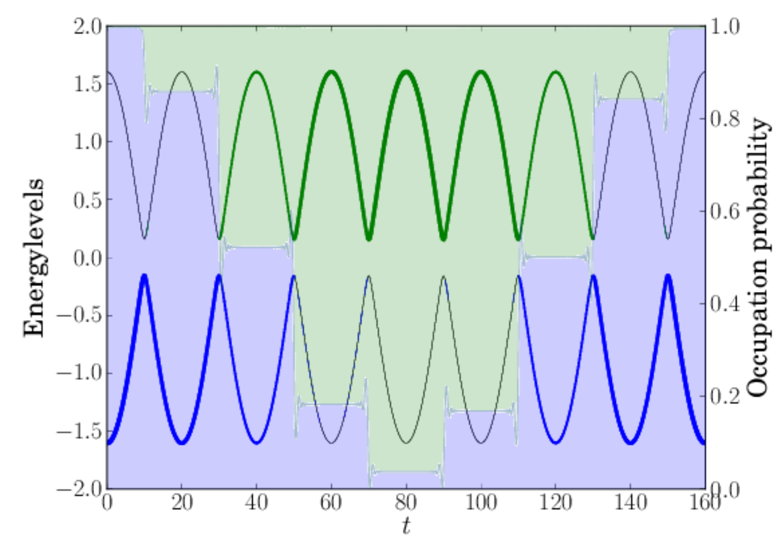
Example: time-dependence

Multiple Landau-Zener transitions

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

```
from qutip import *

# Parameters
epsilon = 0.0
delta = 1.0

# Initial state: start in ground state
psi0 = basis(2,0)

# Hamiltonian
H0 = - delta * sigmaz() - epsilon * sigmax()
H1 = - sigmaz()
h_t = [H0, [H1, 'A * cos(w*t)']]
args = {'A': 10.017, 'w': 0.025*2*pi}

# No dissipation
c_ops = []

# Expectation values
e_ops = [sigmax(), sigmay(), sigmaz()]

# Evolve the system
tlist = linspace(0, 160, 500)
output = mesolve(h_t, psi0, tlist, c_ops, e_ops, args)

# Process and plot result
# ...
```





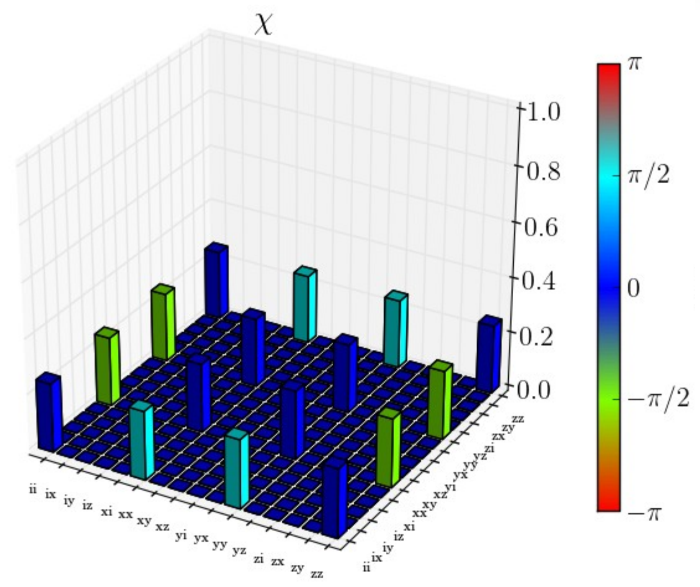
Example: open quantum system

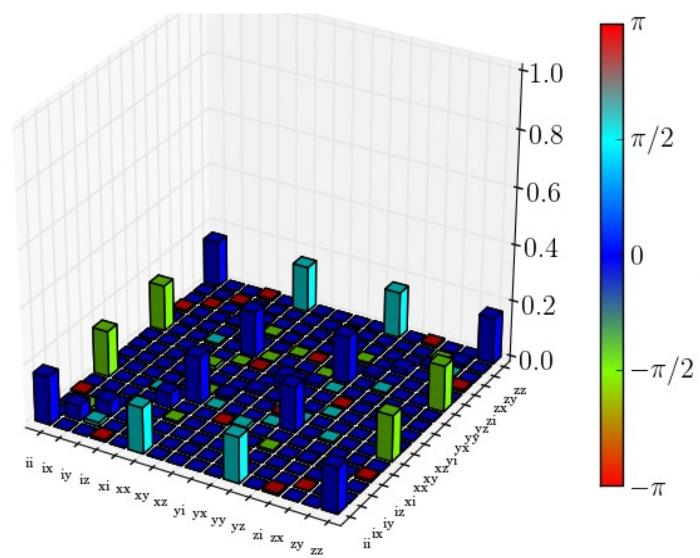
Dissipative two-qubit iSWAP gate

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

```
from qutip import *

g = 1.0 * 2 * pi # coupling strength
g1 = 0.75        # relaxation rate
g2 = 0.25        # dephasing rate
n_th = 1.5       # environment temperature
T = pi/(4*g)

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

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)

U = propagator(H, T, c_ops)

qpt_plot(qpt(U, op_basis), op_labels)
```

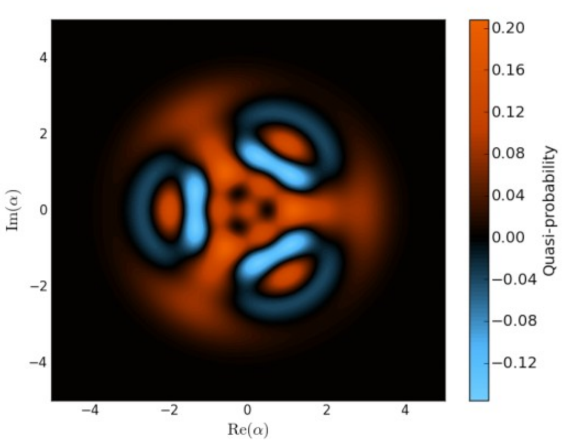
Collapse operators

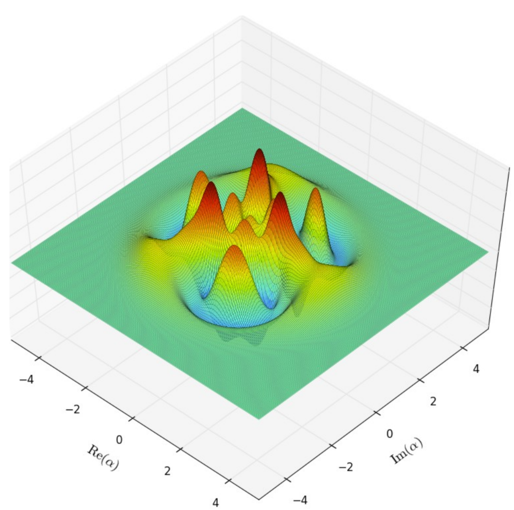


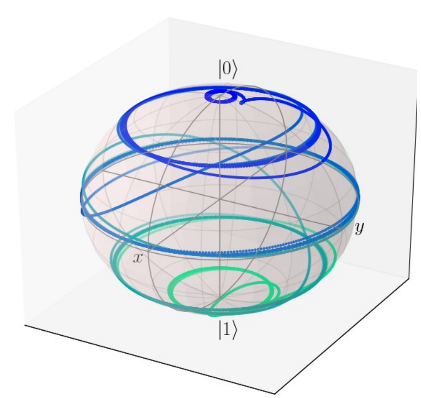


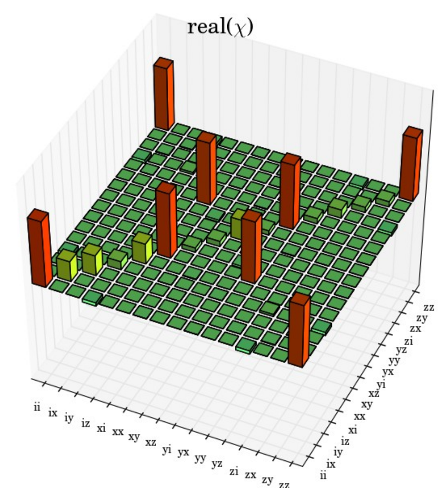
Visualization

- Objectives of visualization in quantum mechanics:
  - Visualize the composition of complex quantum states (superpositions and statistical mixtures).
  - Distinguish between quantum and classical states. Example: Wigner function.
- In QuTiP:
  - Wigner and Q functions, Bloch spheres, process tomography, ...
  - *most common visualization techniques used in quantum mechanics are implemented*









Case-studies in circuit-QED

- IPython notebooks:
  - Jaynes-Cumming-like models
    - Vacuum Rabi oscillations
    - Qubit-gates using a resonators as a bus
    - Single-atom laser
  - Dicke model / Ultrastrong coupling
  - Correlation functions and nonclassicality tests
  - Parametric amplifiers
- Available for download from github:

http://github.com/jrjohansson/qutip-lectures

QuTiP2

The Quantum Toolbox in Python

Summary

More information at:
http://qutip.googlecode.com

- QuTiP: framework for numerical simulations of quantum systems
  - Generic framework for representing quantum states and operators
  - Large number of dynamics solvers
- Main strengths:
  - Ease of use: complex quantum systems can programmed rapidly and intuitively
  - Flexibility: Can be used to solve a wide variety of problems
  - Performance: Near C-code performance due to use of Cython for time-critical functions
- Future developments:
  - Stochastic master equations?
    Non-markovian master equations?