

# An Introduction to Python and Qutip

Peter Kirton

October 16, 2019

- Files at <https://github.com/peterkirton/qutipdemo>

# Setting up

- Files at <https://github.com/peterkirton/qutipdemo>
- Run `ipython` (or `python`) to start a python session
- Scripts can be run either using
  - `run script_name` from `ipython` (better)
  - `python script_name` from `bash`
- `import numpy as np`
- `from scipy.foo import bar`

- Provide large set of library functions for scientific applications
- Linear algebra
- Differential equations
- Optimization
- Fourier transforms etc, etc.

## Quick examples 1: Linear algebra

- `A = np.array([[1, 2], [3, 4]])`
- `val, vec = linalg.eig(A)`

# Quick examples 1: Linear algebra

- `A = np.array([[1, 2], [3, 4]])`
- `val, vec = linalg.eig(A)`
- `B = np.array([[1,2,3],[4,5,6]])`
- `U,s,Vh = linalg.svd(B)`
- Also routines for sparse matrices etc.

## Quick examples 2: ODE

- Use ode class from `scipy.integrate`
- need to specify function of differentials
- Solve:

$$\frac{d^2y}{dx^2} = -5y + J_0(x)$$

## Quick examples 2: ODE

- Use ode class from `scipy.integrate`
- need to specify function of differentials
- Solve:

$$\frac{d^2y}{dx^2} = -5y + J_0(x)$$

- Split into 1st order:

$$\frac{dv}{dx} = -5y + J_0(x)$$

$$\frac{dy}{dx} = v$$



- A Python toolbox for simulating open quantum systems
- (Relatively) simple to use
- Wraps up useful Numpy/Scipy routines
- Efficient
- Easy to set up complex Hilbert/Liouville spaces
- `from qutip import *`
  - Brings all qutip functions into the current namespace
  - Probably bad practice (namespace pollution) but okay for simple scripts

- `fock( $N, m$ )`  $N$  number of basis states,  $m$  Fock state
- `fock_dm( $N, m$ )`
- `coherent( $N, \alpha$ )`  $\alpha$  displacement
- `coherent_dm( $N, \alpha$ )`
- `thermal_dm( $N, n$ )`  $n$  thermal occupation number

- `qeye(N)`
- `create(N)`
- `num(N)`
- `displace(N,  $\alpha$ )`
- `squeeze(N, sp)` `sp` squeezing parameter
- `sigmax()`, `sigmap()` etc
- `sigmap()  $\neq$  create(2)`
- Spin states defined so that `fock(2,1) =  $|\downarrow\rangle = |1\rangle$`

- Append to an object of class Qobj e.g.  
`create(5).dag()=destroy(5)`
- `.dag()`
- `.eigenstates()`
- `.groundstate()`
- `.tr()`

# Some things try

- `vac = fock(5,0)`
- `a = create(5)`
- `a*vac`
- `(a**4)*vac`
- `a.dag()*vac`

# Two spin model

- Two coupled spins

$$H = \Omega_1 \sigma_x^1 + \Omega_2 \sigma_x^2 + g(\sigma_+^1 \sigma_-^2 + \sigma_-^1 \sigma_+^2)$$

$$\dot{\rho} = -i[H, \rho] + \gamma_1 \mathcal{L}[\sigma_-^1] + \gamma_2 \mathcal{L}[\sigma_-^2]$$

- All operators need to be in tensor product space
- $\sigma_x^1 = \text{tensor}(\text{sigmax}(), \text{Is})$
- $\sigma_+^1 \sigma_-^2 = \text{tensor}(\text{sigmap}(), \text{sigmam}())$
- decay is a **list**

- Coupled spin-photon

$$H = \Omega_0 \sigma_z + \Omega_c a^\dagger a + g \sigma_x (a + a^\dagger)$$

$$\dot{\rho} = -i[H, \rho] + \kappa \mathcal{L}[a]$$

- Need to truncate photon Hilbert space  $N_{\text{phot}} = 10$
- Check convergence with this

- Hamiltonian:

$$H = J \sum_i a_i^\dagger a_{i+1} + h.c.$$

$$\dot{\rho} = -i[H, \rho] + \Gamma \sum_i \mathcal{L}[a_i]$$

- Restrict to certain excitation number subspace
- Use `enr_destroy` to create  $a_i$  operators



# Other Useful Things

- `steadystate` - steady state density matrix (or expectation values) of superoperator (or Hamiltonian, decay pair)
- `correlation_ss` - ss two-time correlation function
- `spectrum_ss`
- `correlation_2op_2t` - non-steady two-time correlation function
- `bloch_redfield_tensor`, `bloch_redfield_solve`
- etc., etc.

# Some things to note

- Python is sensitive to whitespace (the end of e.g. for loops is defined by whitespace)

# Some things to note

- Python is sensitive to whitespace (the end of e.g. for loops is defined by whitespace)
- If  $y$  is a **list** then  $x=y$  means  $x$  is a **pointer** to the same object as  $y$ . It will **not** create a copy of  $y$ . Use  $x=\text{copy}(y)$  from `numpy` instead

# Some things to note

- Python is sensitive to whitespace (the end of e.g. for loops is defined by whitespace)
- If  $y$  is a **list** then  $x=y$  means  $x$  is a **pointer** to the same object as  $y$ . It will **not** create a copy of  $y$ . Use  $x=\text{copy}(y)$  from `numpy` instead
- Python is **not** C

# Some things to note

- Python is sensitive to whitespace (the end of e.g. for loops is defined by whitespace)
- If `y` is a **list** then `x=y` means `x` is a **pointer** to the same object as `y`. It will **not** create a copy of `y`. Use `x=copy(y)` from `numpy` instead
- Python is **not** C
- Use the features:
  - Iterables: `enumerate` rather than `range`
  - `in` is really useful
  - Collections module has really useful datatypes
  - Avoid temp variables `a, b = b, a`
  - `try:` Except blocks
- `import this`

- Web:
  - <http://qutip.org/>
  - <http://python.org/>
  - <https://www.scipy.org/>
- Books
  - Learn Python the hard way
  - A Primer on Scientific Programming with Python