example-bloch-redfield

June 25, 2014

1 QuTiP example: Bloch-Redfield Master Equation

J.R. Johansson and P.D. Nation

For more information about QuTiP see http://qutip.org

In [1]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [2]:
```
from qutip import *
```

1.1 Single qubit dynamics

In [3]:
```
def qubit_integrate(w, theta, gamma1, gamma2, psi0, tlist):
    # Hamiltonian
    sx = sigmax()
    sy = sigmay()
    sz = sigmaz()
    sm = sigmam()
    H = w * (cos(theta) * sz + sin(theta) * sx)

    # Lindblad master equation
    c_op_list = []
    n_th = 0.0 # zero temperature
    rate = gamma1 * (n_th + 1)
    if rate > 0.0:
        c_op_list.append(sqrt(rate) * sm)
    rate = gamma1 * n_th
    if rate > 0.0:
        c_op_list.append(sqrt(rate) * sm.dag())
    lme_results = mesolve(H, psi0, tlist, c_op_list, [sx, sy, sz]).expect

    # Bloch-Redfield tensor
    #ohmic_spectrum = lambda w: gamma1 * w / (2*pi)**2 * (w > 0.0)
    def ohmic_spectrum(w):
        if w == 0.0:
            # dephasing inducing noise
            return gamma1/2
        else:
            # relaxation inducing noise
            return gamma1/2 * w / (2*pi) * (w > 0.0)


    brme_results = brmesolve(H, psi0, tlist, [sx], [sx, sy, sz], [ohmic_spectrum]).expect
```

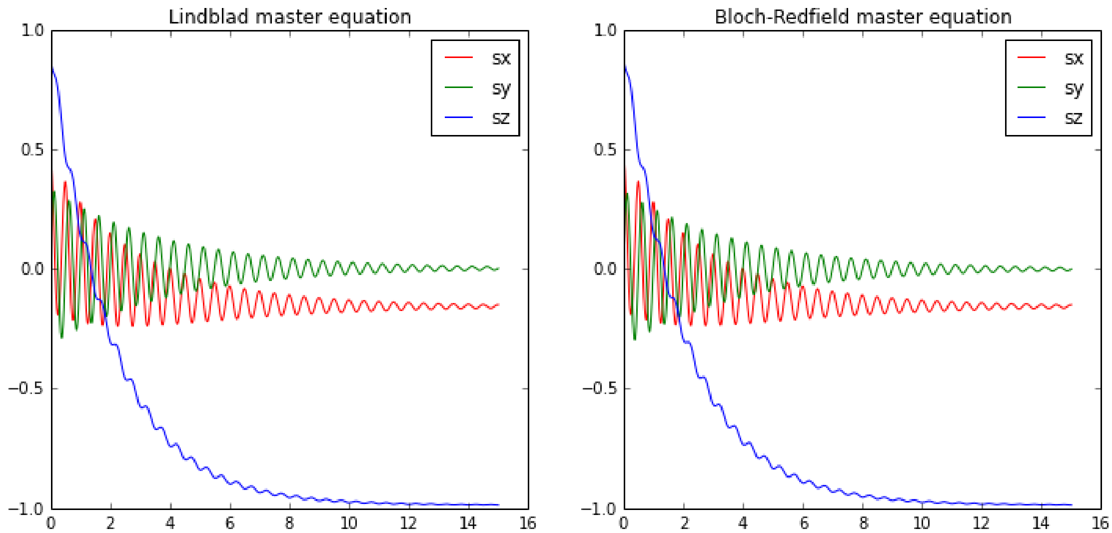
```
    # alternative:
    #R, ekets = bloch_redfield_tensor(H, [sx], [ohmic_spectrum])
    #brme_results = bloch_redfield_solve(R, ekets, psi0, tlist, [sx, sy, sz])

    return lme_results, brme_results
```

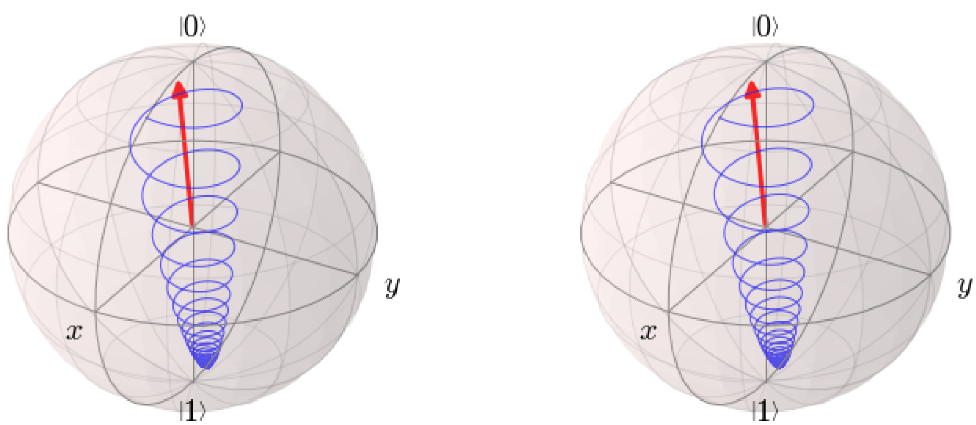
In [4]:
```
w     = 1.0 * 2 * pi  # qubit angular frequency
theta = 0.05 * pi     # qubit angle from sigma_z axis (toward sigma_x axis)
gamma1 = 0.5          # qubit relaxation rate
gamma2 = 0.0          # qubit dephasing rate
# initial state
a = 0.8
psi0 = (a* basis(2,0) + (1-a)*basis(2,1))/(sqrt(a**2 + (1-a)**2))
tlist = linspace(0,15,5000)
```

In [5]:
```
lme_results, brme_results = qubit_integrate(w, theta, gamma1, gamma2, psi0, tlist)
```

In [6]:
```
fig = figure(figsize=(12,12))
ax = fig.add_subplot(2,2,1)
title('Lindblad master equation')
ax.plot(tlist, lme_results[0], 'r')
ax.plot(tlist, lme_results[1], 'g')
ax.plot(tlist, lme_results[2], 'b')
ax.legend(("sx", "sy", "sz"))

ax = fig.add_subplot(2,2,2)
title('Bloch-Redfield master equation')
ax.plot(tlist, brme_results[0], 'r')
ax.plot(tlist, brme_results[1], 'g')
ax.plot(tlist, brme_results[2], 'b')
ax.legend(("sx", "sy", "sz"))


sphere=Bloch(axes=fig.add_subplot(2,2,3, projection='3d'))
sphere.add_points([lme_results[0],lme_results[1],lme_results[2]], meth='l')
sphere.vector_color = ['r']
sphere.add_vectors([sin(theta),0,cos(theta)])
sphere.make_sphere()

sphere=Bloch(axes=fig.add_subplot(2,2,4, projection='3d'))
sphere.add_points([brme_results[0],brme_results[1],brme_results[2]], meth='l')
sphere.vector_color = ['r']
sphere.add_vectors([sin(theta),0,cos(theta)])
sphere.make_sphere()
```

```
/usr/lib/python3/dist-packages/numpy/core/numeric.py:460: ComplexWarning: Casting complex values to real
  return array(a, dtype, copy=False, order=order)
```






```
<matplotlib.figure.Figure at 0x7fa5680cffd0>
```

```
<matplotlib.figure.Figure at 0x7fa5680cff98>
```

1.2 Coupled qubits

```
In [7]: def qubit_integrate(w, theta, g, gamma1, gamma2, psi0, tlist):
            #
            # Hamiltonian
            #
            sx1 = tensor(sigmax(),qeye(2))
            sy1 = tensor(sigmay(),qeye(2))
            sz1 = tensor(sigmaz(),qeye(2))
            sm1 = tensor(sigmam(),qeye(2))
```

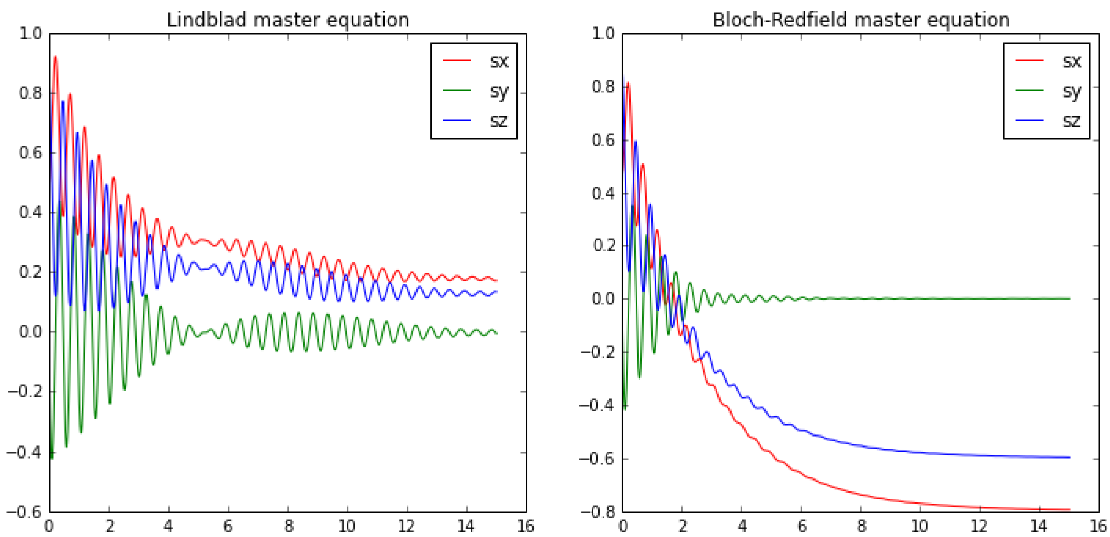
```
    sx2 = tensor(qeye(2),sigmax())
    sy2 = tensor(qeye(2),sigmay())
    sz2 = tensor(qeye(2),sigmaz())
    sm2 = tensor(qeye(2),sigmam())

    H  = w[0] * (cos(theta[0]) * sz1 + sin(theta[0]) * sx1) # qubit 1
    H += w[1] * (cos(theta[1]) * sz2 + sin(theta[1]) * sx2) # qubit 2
    H += g * sx1 * sx2                                      # interaction

    #
    # Lindblad master equation
    #
    c_op_list = []
    n_th = 0.0 # zero temperature
    rate = gamma1[0] * (n_th + 1)
    if rate > 0.0: c_op_list.append(sqrt(rate) * sm1)
    rate = gamma1[1] * (n_th + 1)
    if rate > 0.0: c_op_list.append(sqrt(rate) * sm2)

    lme_results = mesolve(H, psi0, tlist, c_op_list, [sx1, sy1, sz1]).expect

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

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

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

    # alternative:
    #R, ekets = bloch_redfield_tensor(H, [sx1, sx2], [ohmic_spectrum1, ohmic_spectrum2])
    #brme_results = brmesolve(R, ekets, psi0, tlist, [sx1, sy1, sz1])

    return lme_results, brme_results
```

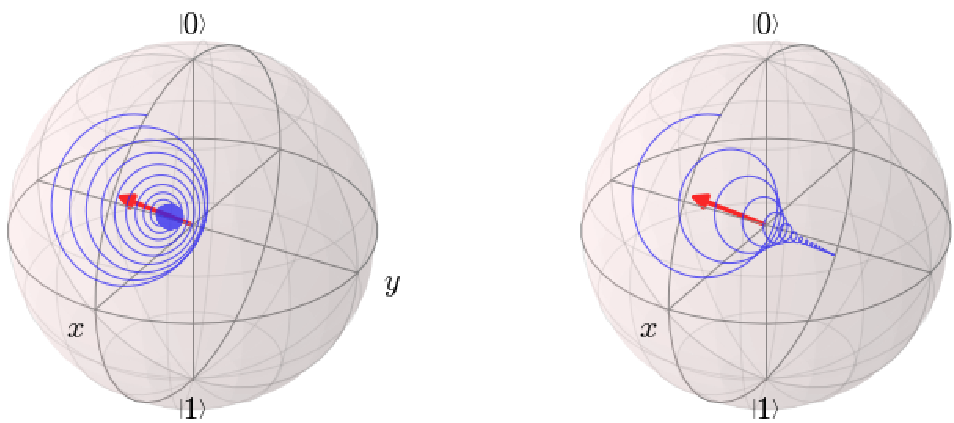
In [8]:
```
w     = array([1.0, 1.0]) * 2 * pi   # qubit angular frequency
theta = array([0.15, 0.45]) * 2 * pi # qubit angle from sigma_z axis (toward sigma_x axis)
gamma1 = [0.25, 0.35]                # qubit relaxation rate
gamma2 = [0.0, 0.0]                  # qubit dephasing rate
```

```
g       = 0.1 * 2 * pi
# initial state
a = 0.8
psi1 = (a*basis(2,0) + (1-a)*basis(2,1))/(sqrt(a**2 + (1-a)**2))
psi2 = ((1-a)*basis(2,0) + a*basis(2,1))/(sqrt(a**2 + (1-a)**2))
psi0 = tensor(psi1, psi2)

tlist = linspace(0,15,5000)
```

In [9]:
```
lme_results, brme_results = qubit_integrate(w, theta, g, gamma1, gamma2, psi0, tlist)
```

In [10]:
```
fig = figure(figsize=(12,12))
ax = fig.add_subplot(2,2,1)
title('Lindblad master equation')
ax.plot(tlist, lme_results[0], 'r')
ax.plot(tlist, lme_results[1], 'g')
ax.plot(tlist, lme_results[2], 'b')
ax.legend(("sx", "sy", "sz"))

ax = fig.add_subplot(2,2,2)
title('Bloch-Redfield master equation')
ax.plot(tlist, brme_results[0], 'r')
ax.plot(tlist, brme_results[1], 'g')
ax.plot(tlist, brme_results[2], 'b')
ax.legend(("sx", "sy", "sz"))

sphere=Bloch(axes=fig.add_subplot(2,2,3, projection='3d'))
sphere.add_points([lme_results[0],lme_results[1],lme_results[2]], meth='l')
sphere.vector_color = ['r']
sphere.add_vectors([sin(theta[0]),0,cos(theta[0])])
sphere.make_sphere()

sphere=Bloch(axes=fig.add_subplot(2,2,4, projection='3d'))
sphere.add_points([brme_results[0],brme_results[1],brme_results[2]], meth='l')
sphere.vector_color = ['r']
sphere.add_vectors([sin(theta[0]),0,cos(theta[0])])
sphere.make_sphere()
```






```
<matplotlib.figure.Figure at 0x7fa568089898>
```

```
<matplotlib.figure.Figure at 0x7fa567a7beb8>
```

1.3 Versions

```
In [11]: from qutip.ipynbtools import version_table

         version_table()

Out[11]: <IPython.core.display.HTML at 0x7fa56750d6d8>
```