QuTiP lecture: Single-Atom-Lasing

Author: J.R. Johansson, robert@riken.jp

http://dml.riken.jp/~rob/

Latest version of this ipython notebook lecture is available at: http://github.com/jrjohansson/qutip-lectures

In [1]:
```
# setup the matplotlib graphics library and configure it to show
# figures inline in the notebook
%pylab inline
```

```
Welcome to pylab, a matplotlib-based Python environment [backend: module://IPython.zmq.pylab.backend_inline].
For more information, type 'help(pylab)'.
```

In [2]:
```
# make qutip available in the rest of the notebook
from qutip import *
```

Introduction and model

Consider a single atom coupled to a single cavity mode. If there atom excitation rate exceeds the relaxation rate, a population inversion can occur in the atom, and if coupled to the cavity the atom can then act as a photon pump on the cavity.

The coherent dynamics in this model is described by the Hamiltonian

$$H = \hbar\omega_0 a^\dagger a + \frac{1}{2}\hbar\omega_a\sigma_z + \hbar g\sigma_x(a^\dagger + a)$$

where ω_0 is the cavity energy splitting, ω_a is the atom energy splitting and g is the atom-cavity interaction strength.

In addition to the coherent dynamics the following incoherent processes are also present: 1) κ relaxation and thermal excitations of the cavity, Γ atomic excitation rate (pumping process).

The Lindblad master equation for the model is:

$$\frac{d}{dt}\rho = -i[H,\rho] + \Gamma\left(\sigma_+\rho\sigma_- - \frac{1}{2}\sigma_-\sigma_+\rho - \frac{1}{2}\rho\sigma_-\sigma_+\right) + \kappa(1+n_{\rm th})\left(a\rho a^\dagger - \frac{1}{2}a^\dagger a\rho - \frac{1}{2}\rho a^\dagger a\right) + \kappa n_{\rm th}\left(a^\dagger\rho a - \frac{1}{2}aa^\dagger\rho - \frac{1}{2}\rho aa^\dagger\right)$$

in units where $\hbar = 1$.

References:

- Yi Mu, C.M. Savage, Phys. Rev. A 46, 5944 (1992)
- D.A. Rodrigues, J. Imbers, A.D. Armour, Phys. Rev. Lett. 98, 067204 (2007)
- S. Ashhab, J.R. Johansson, A.M. Zagoskin, F. Nori, New J. Phys. 11, 023030 (2009)

Problem parameters

In [3]:
```
w0 = 1.0  * 2 * pi  # cavity frequency
wa = 1.0  * 2 * pi  # atom frequency
g  = 0.05 * 2 * pi  # coupling strength
```

```
kappa = 0.03      # cavity dissipation rate
gamma = 0.00       # atom dissipation rate
Gamma = 0.4        # atom pump rate

N = 60             # number of cavity fock states
n_th_a = 0.0       # avg number of thermal bath excitation

tlist = linspace(0, 150, 101)
```

Setup the operators, the Hamiltonian and initial state

In [4]:
```
# intial state
psi0 = tensor(basis(N,0), basis(2,0)) # start without excitations

# operators
a  = tensor(destroy(N), qeye(2))
sm = tensor(qeye(N), destroy(2))
sx = tensor(qeye(N), sigmax())

# Hamiltonian
H = w0 * a.dag() * a + wa * sm.dag() * sm + g * (a.dag() + a) * sx
```

In [5]:
```
H
```

Out [5]: Quantum object: dims = [[60, 2], [60, 2]], shape = [120, 120], type = oper, isHerm = True

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.314159265359 & 0.0 & \cdots & 0.0 & 0.0 & 0.0 \\ 0.0 & 6.28318530718 & 0.314159265359 & 0.0 & 0.0 & \cdots & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.314159265359 & 6.28318530718 & 0.0 & 0.0 & \cdots & 0.0 & 0.0 & 0.0 \\ 0.314159265359 & 0.0 & 0.0 & 12.5663706144 & 0.444288293816 & \cdots & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.444288293816 & 12.5663706144 & \cdots & 0.0 & 0.0 & 0.0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & \cdots & 364.424747816 & 2.39256568408 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & \cdots & 2.39256568408 & 364.424747816 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & \cdots & 0.0 & 0.0 & 370.707933124 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & \cdots & 0.0 & 0.0 & 2.41310310527 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & \cdots & 0.0 & 2.41310310527 & 0.0 \end{pmatrix}$$

Create a list of collapse operators that describe the dissipation

In [6]:
```
# collapse operators
c_ops = []

rate = kappa * (1 + n_th_a)
if rate > 0.0:
    c_ops.append(sqrt(rate) * a)

rate = kappa * n_th_a
if rate > 0.0:
    c_ops.append(sqrt(rate) * a.dag())

rate = gamma
if rate > 0.0:
    c_ops.append(sqrt(rate) * sm)

rate = Gamma
if rate > 0.0:
    c_ops.append(sqrt(rate) * sm.dag())
```

Evolve the system

Here we evolve the system with the Lindblad master equation solver, and we request that the expectation values of the operators $a^\dagger a$ and $\sigma_+\sigma_-$ are returned by the solver by passing the list `[a.dag()*a, sm.dag()*sm]` as the fifth argument to the solver.

In [7]:
```
opt = Odeoptions(nsteps=2000) # allow extra time-steps
output = mesolve(H, psi0, tlist, c_ops, [a.dag() * a, sm.dag() * sm], options=opt)
```

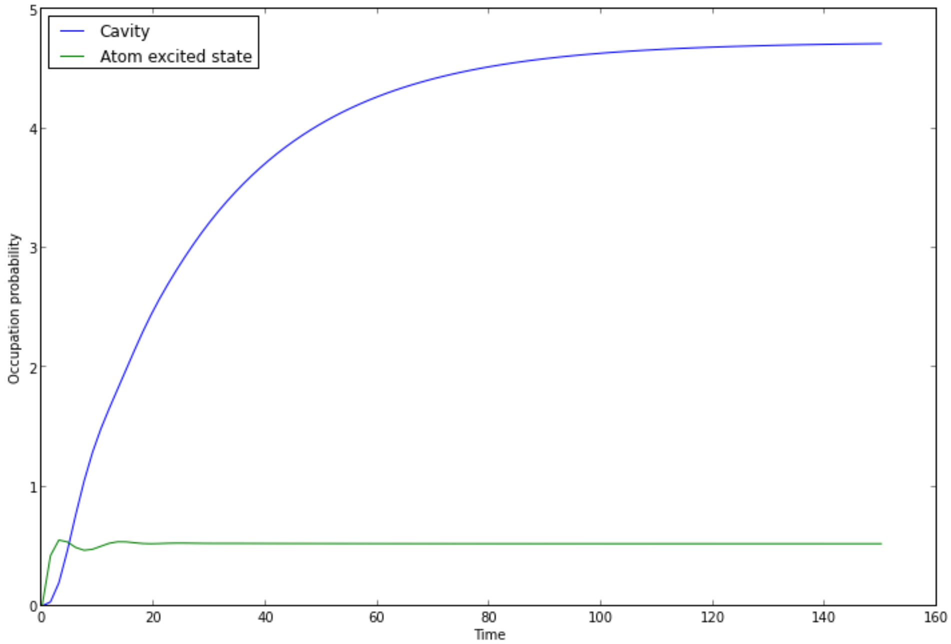
Visualize the results

Here we plot the excitation probabilities of the cavity and the atom (these expectation values were calculated by the `mesolve` above).

In [8]:
```
n_c = output.expect[0]
n_a = output.expect[1]

fig, axes = subplots(1, 1, sharex=True, figsize=(12,8))
```

```
axes.plot(tlist, n_c, label="Cavity")
axes.plot(tlist, n_a, label="Atom excited state")
axes.legend(loc=0)
axes.set_xlabel('Time')
axes.set_ylabel('Occupation probability')
```

Out [8]: <matplotlib.text.Text at 0x4166f90>

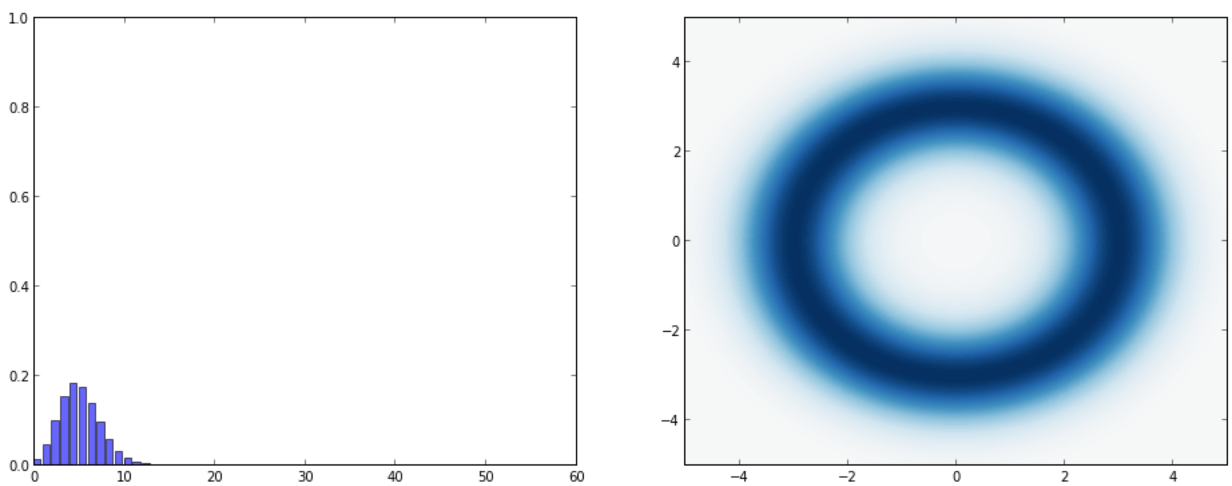


Steady state: cavity fock-state distribution and wigner function

In [9]:
```
rho_ss = steadystate(H, c_ops)
```

In [10]:
```
fig, axes = subplots(1, 2, figsize=(16,6))

xvec = linspace(-5,5,200)

rho_cavity = ptrace(rho_ss, 0)
W = wigner(rho_cavity, xvec, xvec)
wlim = abs(W).max()
axes[1].contourf(xvec, xvec, W, 100, norm=mpl.colors.Normalize(-wlim,wlim), cmap=get_cmap('RdBu'))

axes[0].bar(arange(0, N), real(rho_cavity.diag()), color="blue", alpha=0.6)
axes[0].set_ylim(0, 1)
axes[0].set_xlim(0, N)

#ax.legend()
#ax.set_xlabel('Time')
#ax.set_ylabel('Occupation probability')
```

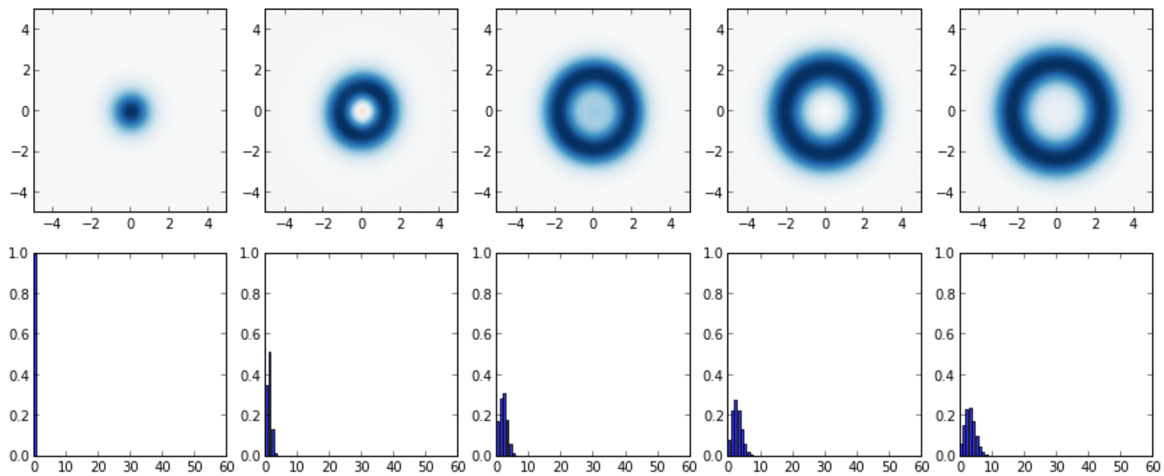
Out [10]: (0, 60)



Cavity fock-state distribution and Wigner function as a function of time

In [11]:
```
opt = Odeoptions(nsteps=5000) # allow extra time-steps
output = mesolve(H, psi0, linspace(0, 25, 5), c_ops, [], options=opt)
```

In [12]:
```
rho_ss_sublist = output.states

xvec = linspace(-5,5,200)

fig, axes = subplots(2, len(rho_ss_sublist), figsize=(3*len(rho_ss_sublist), 6))

for idx, rho_ss in enumerate(rho_ss_sublist):

    # trace out the cavity density matrix
    rho_ss_cavity = ptrace(rho_ss, 0)

    # calculate its wigner function
    W = wigner(rho_ss_cavity, xvec, xvec)

    # plot its wigner function
    wlim = abs(W).max()
    axes[0,idx].contourf(xvec, xvec, W, 100, norm=mpl.colors.Normalize(-wlim,wlim), cmap=get_cmap('RdBu'))

    # plot its fock-state distribution
    axes[1,idx].bar(arange(0, N), real(rho_ss_cavity.diag()), color="blue", alpha=0.8)
    axes[1,idx].set_ylim(0, 1)
    axes[1,idx].set_xlim(0, N)
```



Steady state average photon occupation in cavity as a function of pump rate

References:

- S. Ashhab, J.R. Johansson, A.M. Zagoskin, F. Nori, New J. Phys. 11, 023030 (2009)

In [13]:
```
def calulcate_avg_photons(N, Gamma):

    # collapse operators
    c_ops = []

    rate = kappa * (1 + n_th_a)
    if rate > 0.0:
        c_ops.append(sqrt(rate) * a)

    rate = kappa * n_th_a
    if rate > 0.0:
        c_ops.append(sqrt(rate) * a.dag())

    rate = gamma
    if rate > 0.0:
        c_ops.append(sqrt(rate) * sm)

    rate = Gamma
    if rate > 0.0:
        c_ops.append(sqrt(rate) * sm.dag())

    # Ground state and steady state for the Hamiltonian: H = H0 + g * H1
    rho_ss = steadystate(H, c_ops)
```

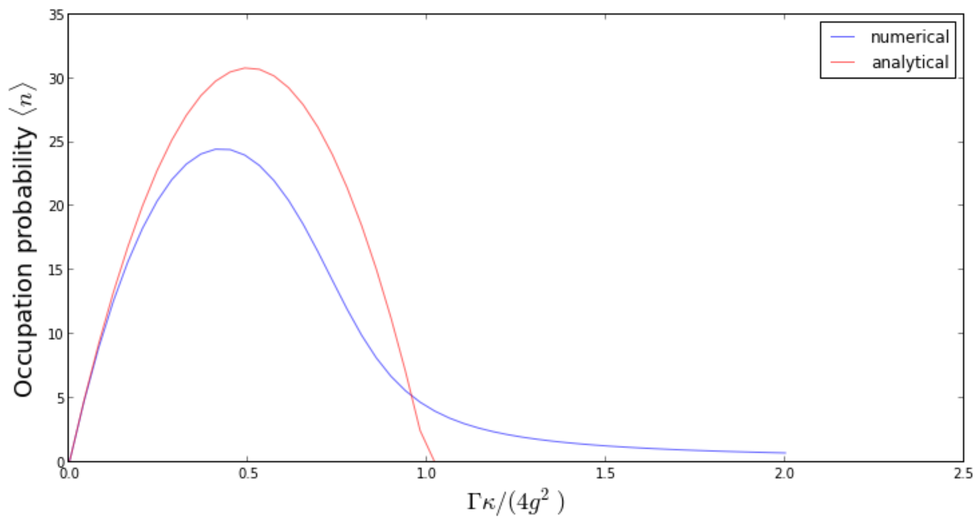
```
    n_cavity = expect(a.dag() * a, rho_ss)

    return n_cavity
```

In [14]:
```
Gamma_max = 2 * (4*g**2) / kappa
Gamma_vec = linspace(0.0, Gamma_max, 50)

n_avg_vec = [calulcate_avg_photons(N, Gamma) for Gamma in Gamma_vec]
```

In [15]:
```
n_avg_analytical_vec = (Gamma_vec / (2 * kappa)) * (1 - Gamma_vec * kappa / (4 * g**2))
n_avg_analytical_vec = n_avg_analytical_vec * (n_avg_analytical_vec > 0)
```

In [16]:
```
fig, axes = subplots(1, 1, figsize=(12,6))

axes.plot(Gamma_vec * kappa / (4*g**2), n_avg_vec, color="blue", alpha=0.6, label="numerical")
axes.plot(Gamma_vec * kappa / (4*g**2), n_avg_analytical_vec, color="red", alpha=0.6, label="analytical")

axes.set_xlabel(r'$\Gamma\kappa/(4g^2)$', fontsize=18)
axes.set_ylabel(r'Occupation probability $\langle n \rangle$', fontsize=18)
axes.legend()
```

Out [16]: <matplotlib.legend.Legend at 0x7286e10>



Here we see that lasing is suppressed for $\Gamma\kappa/(4g^2) > 1$.

Let's look at the fock-state distribution at $\Gamma\kappa/(4g^2) = 0.5$ (lasing regime) and $\Gamma\kappa/(4g^2) = 1.5$ (suppressed regime):

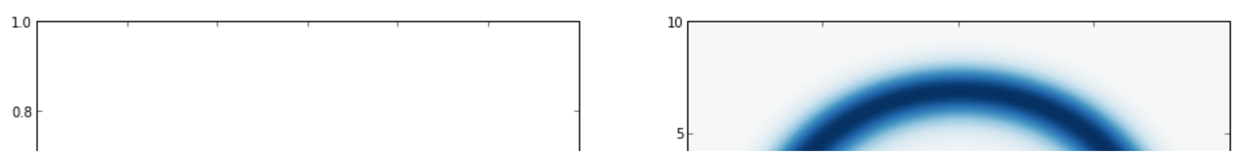
Case 1: $\Gamma\kappa/(4g^2) = 0.5$

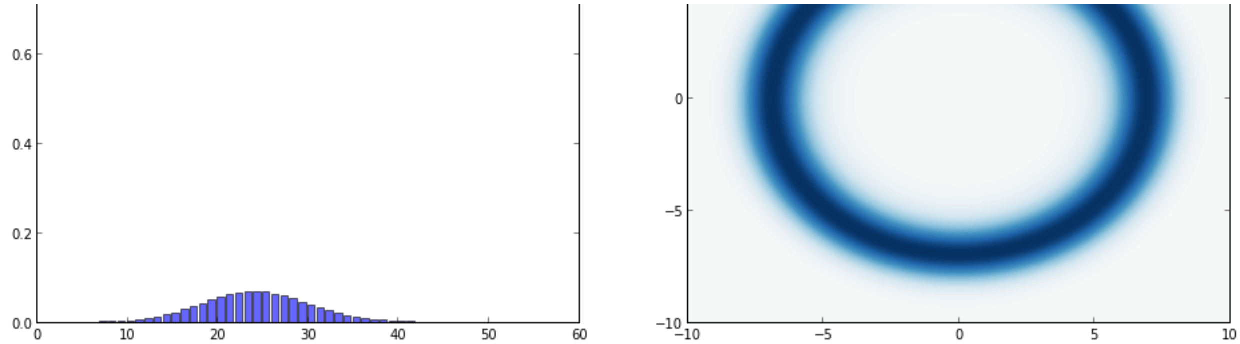
In [17]:
```
Gamma = 0.5 * (4*g**2) / kappa
```

In [18]:
```
c_ops = [sqrt(kappa * (1 + n_th_a)) * a, sqrt(kappa * n_th_a) * a.dag(), sqrt(gamma) * sm, sqrt(Gamma) * sm.dag()]

rho_ss = steadystate(H, c_ops)
```

In [19]:
```
fig, axes = subplots(1, 2, figsize=(16,6))

xvec = linspace(-10,10,200)

rho_cavity = ptrace(rho_ss, 0)
W = wigner(rho_cavity, xvec, xvec)
wlim = abs(W).max()
axes[1].contourf(xvec, xvec, W, 100, norm=mpl.colors.Normalize(-wlim,wlim), cmap=get_cmap('RdBu'))

axes[0].bar(arange(0, N), real(rho_cavity.diag()), color="blue", alpha=0.6)
axes[0].set_ylim(0, 1)
axes[0].set_xlim(0, N)
```

Out [19]: (0, 60)





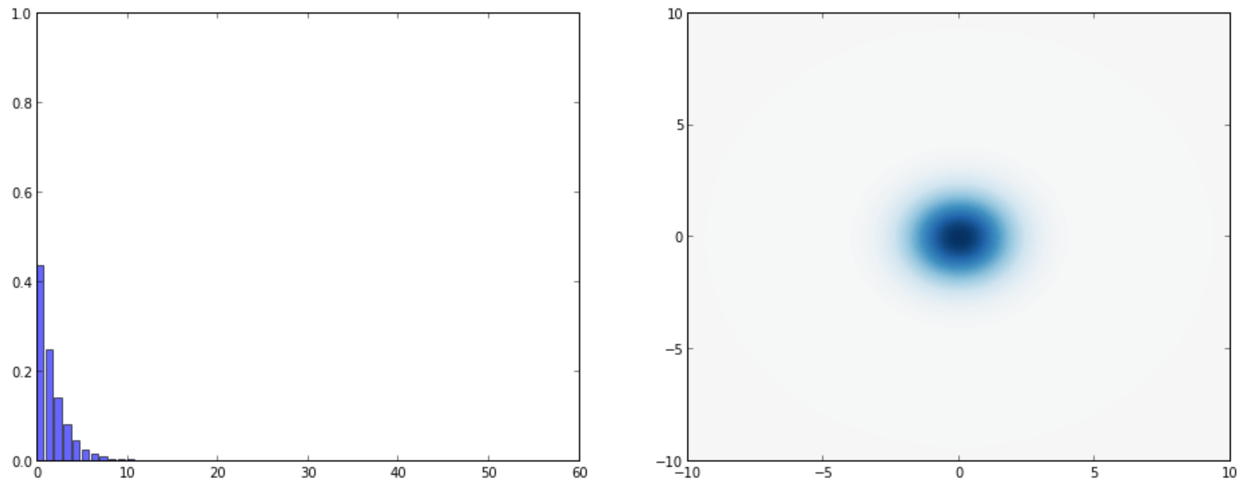
Case 2: $\Gamma\kappa/(4g^2) = 1.5$

In [20]:
```
Gamma = 1.5 * (4*g**2) / kappa
```

In [21]:
```
c_ops = [sqrt(kappa * (1 + n_th_a)) * a, sqrt(kappa * n_th_a) * a.dag(), sqrt(gamma) * sm, sqrt(Gamma) * sm.dag()]

rho_ss = steadystate(H, c_ops)
```

In [22]:
```
fig, axes = subplots(1, 2, figsize=(16,6))

xvec = linspace(-10,10,200)

rho_cavity = ptrace(rho_ss, 0)
W = wigner(rho_cavity, xvec, xvec)
wlim = abs(W).max()
axes[1].contourf(xvec, xvec, W, 100, norm=mpl.colors.Normalize(-wlim,wlim), cmap=get_cmap('RdBu'))

axes[0].bar(arange(0, N), real(rho_cavity.diag()), color="blue", alpha=0.6)
axes[0].set_ylim(0, 1)
axes[0].set_xlim(0, N)
```

Out [22]: (0, 60)



Too large pumping rate Γ kills the lasing process: reversed threshold.