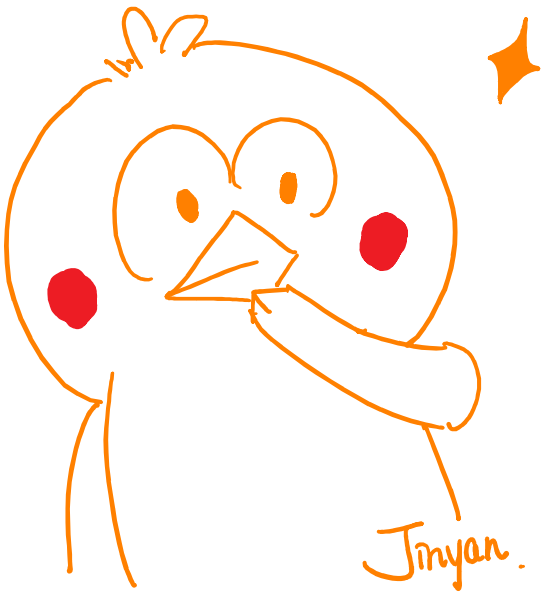
Homework 2

Physics 542 - Quantum Optics
Professor Alex Kuzmich



Jinyan Miao

Fall 2023

1

Here we denote the probability of the atom being in state $|2\rangle$ after the pulse is over but state $|2\rangle$ has not had time to decay via spontaneous emission as $\langle P_2 \rangle$. It is reasonable to assume that

$$\langle P_2 \rangle = f(\gamma, \delta, \Omega, \tau)\,.$$

We first note that, without spontaneous decay ($\gamma = 0$), and given that the pulse is off-resonant, $\langle P_2 \rangle = 0$ under adiabatic approximation [as discussed in Section 2.6.1 on Berman]. Thus having $\langle P_2 \rangle \neq 0$ relies on the occurrence of spontaneous decay from $|2\rangle$ to $|1\rangle$ during the pulse interacting with the atom. The spontaneous decay from $|2\rangle$ to $|1\rangle$ allows the atom to *jump* between the two dressed state which is otherwise prohibited in the adiabatic limit when the pulse is off-resonant.

Thus, when the interaction between the pulse and the atom is on, and $|\delta|$ is large, the time-averaged population of $|2\rangle$ can be approximated [from Eq. (2.93) on Berman]

$$\mathbb{P}_2 = \overline{|c_2(t)|^2} = \approx \frac{|\Omega_0|^2}{\delta^2}\,,$$

and thus the probability of spontaneous decay occurrence is

$$\mathbb{P}_{\text{decay}} \approx \frac{|\Omega_0|^2}{\delta^2}\gamma\tau\,.$$

As we want the atom to be in state $|2\rangle$ when the pulse is over, we require the atom in $|1\rangle$ after the spontaneous decay to evolve to state $|2\rangle$ again while interacting with the pulse, thus we append another factor of $\mathbb{P}_2$, and the final result is

$$\langle P_2 \rangle \sim \mathbb{P}_2 \cdot \mathbb{P}_{\text{decay}} = \frac{|\Omega_0|^2}{\delta^2}\frac{|\Omega_0|^2}{\delta^2}\gamma\tau = \frac{|\Omega_0|^4\gamma\tau}{\delta^4}\,.$$

As we see here, the larger the detuning δ, the smaller the probability $\langle P_2 \rangle$. This makes sense because the larger the detuning the smaller the probability of the atom in state $|2\rangle$ when interacting with the pulse.

2

Here we consider the real variables

$$u = \widetilde{\rho}_{12} + \widetilde{\rho}_{21}\,, \qquad v = i(\widetilde{\rho}_{12} - \widetilde{\rho}_{21})\,, \qquad w = \widetilde{\rho}_{22} - \widetilde{\rho}_{11}\,, \qquad m = \widetilde{\rho}_{22} - \widetilde{\rho}_{11}\,,$$

where $\widetilde{\rho}_{ij}$ is the density matrix elements in the field interaction representation. Writing $\mathbf{B} = (u, v, w)$, and $\mathbf{\Omega}(t) = (|\Omega_0(t)|,\, 0,\, \delta(t))$, without the relaxation, the dynamic of the system reads [Eq. (3.74) from Berman]

$$\frac{d\mathbf{B}}{dt} = \mathbf{\Omega}(t) \times \mathbf{B}\,.$$

At $t = -\infty$, it is obvious that we have $\mathbf{B} = (0, 0, 1)$, and $\mathbf{\Omega}(t) \approx (0, 0, \delta(t))$. In the limit where $\chi/\delta \ll 1$ and $\dot{\chi}/(\chi\delta) \ll 1$, we know that $\Omega(t) = (|\Omega_0|^2 + \delta^2)^{1/2}$ is sufficiently large to ensure that the Block vector $\mathbf{B}$ precesses many times around $\mathbf{\Omega}$ before $\mathbf{\Omega}$ rotates. As time increases, $\mathbf{\Omega}$ rotates sufficiently slowly in the wu-plane, and the Bloch vector adiabatically follows the rotation of $\mathbf{\Omega}$. Thus one can compute [Equations 3.93 from Berman]

$$w(t) = -\cos(2\theta(t))\,, \qquad v(t) = 0\,, \qquad u(t) = -\sin(2\theta(t))\,,$$

from which we obtain

$$\rho_{22}(t) = \sin^2(\theta(t))\,,$$

where we have

$$\tan(2\theta) = \left|\frac{\Omega_0}{\delta}\right|\,.$$

Notice further that, as we have assumed $\chi/\delta \ll 1$, thus we can have the approximation

$$\left|\frac{\Omega_0}{\delta}\right| = \left|\frac{2\chi}{\delta}\right| = \tan(2\theta) \approx 2\theta\,,$$

and thus

$$\left|\frac{\chi}{\delta}\right|^2 \approx \theta^2 \approx \sin^2(\theta) = \rho_{22}(t)\,,$$

as expected.

3

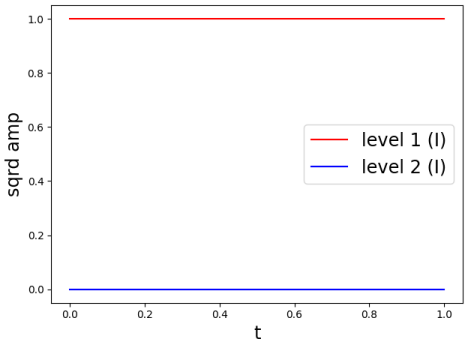
Script for P3

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

def normalize(wf):
    normalization = np.sqrt(wf[0]**2 + wf[1]**2)
    return wf/normalization

def evolve_wavefunction_dt(wf_init, wf, Gamma, t_old, t_new):
    alpha_0 = wf_init[0]
    beta_0 = wf_init[1]
    alpha_old = wf[0]
    beta_old = wf[1]
    if beta_old == 0:
        return wf
    else:
        dp = Gamma*(abs(beta_old)**2)*(t_new-t_old)
        epsilon = np.random.random()
        if dp > epsilon:
            wf = np.array([1,0])
            return wf
        else:
            denominator = np.sqrt(abs(alpha_0)**2+abs(beta_0)**2*np.exp(-
    Gamma*t_new))
            alpha_new = alpha_0/denominator
            beta_new = beta_0*np.exp(-Gamma*t_new/2)/denominator
            wf = normalize(np.array([alpha_new,beta_new]))
            return wf

def evolve_wavefunction_finite(t_arr, Gamma, Psi_init):
    wf_evolved = [Psi_init]
    for i in range(len(t_arr)-1):
        t_old = t_arr[i]+t_arr[0]
        t_new = t_arr[i+1]+t_arr[0]
        wf_old = wf_evolved[-1]
        wf_new = evolve_wavefunction_dt(Psi_init, wf_old, Gamma, t_old,
    t_new)
        wf_evolved.append(wf_new)
    return wf_evolved

def average_over_realizations(t_arr, Gamma, N, Psi_init):
    N_wf_sq_array = []
    for n in range(N):
        wf_evolved = evolve_wavefunction_finite(t_arr, Gamma, Psi_init)
        wf_evolved_sq = np.array(wf_evolved)**2
```

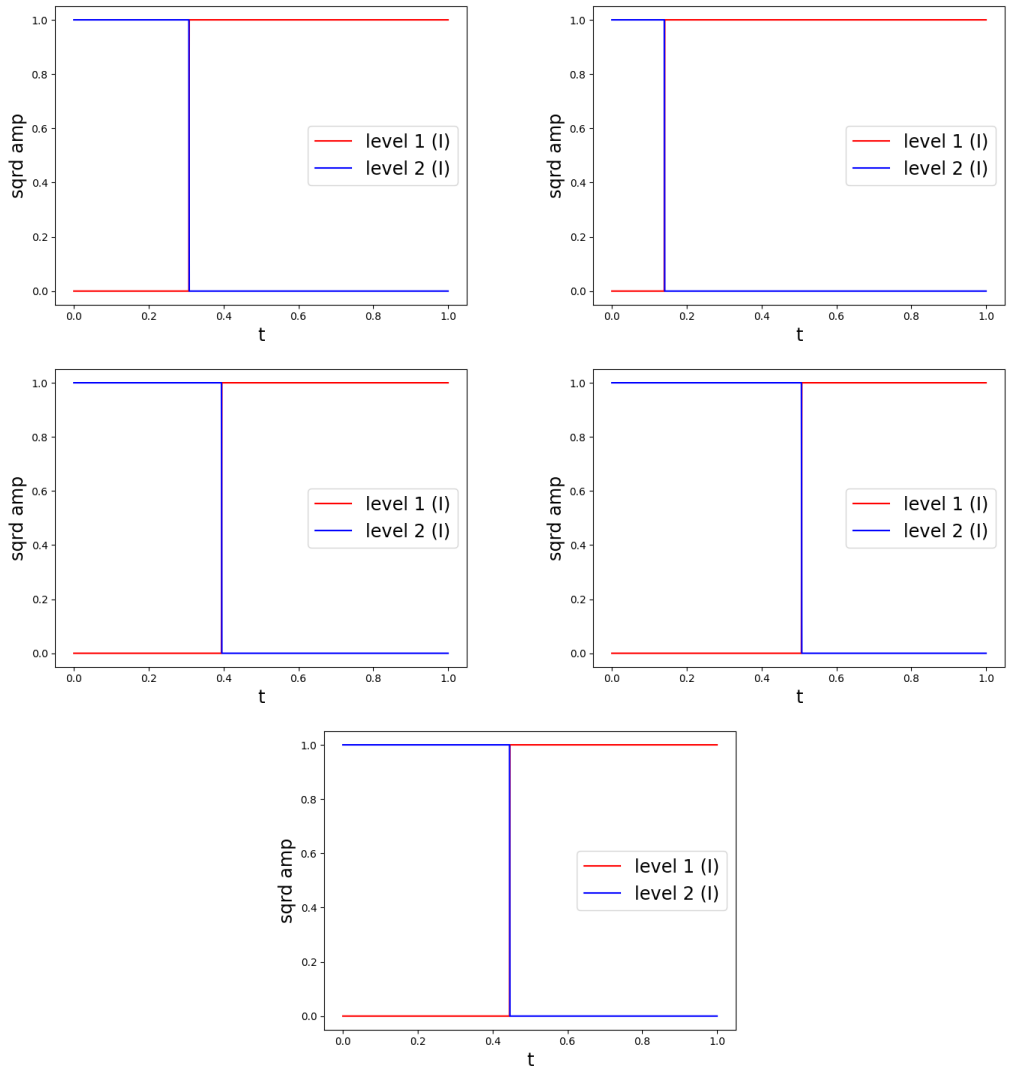
```
        N_wf_sq_array.append(wf_evolved_sq)
    wf_sq_averaged = np.mean(np.array(N_wf_sq_array), axis=0)
    return wf_sq_averaged

def exponential(t, Gamma, amp):
    return np.exp(-t*Gamma)*amp


Gamma = 5 # decay rate
dt = 0.001 # the time jump
t = Gamma/5. # the final time
t_arr = np.linspace(0,t,int(t/dt)) # the values of t in the time interval
N = 500 # number of realizations to average over

# first wavefunction
Unnormalized_Psi_init1 = np.array([0,1])
Unnormalized_Psi_init2 = np.array([1,1])
Psi_init1 = normalize(Unnormalized_Psi_init1)
Psi_init2 = normalize(Unnormalized_Psi_init2)

wf_sq_averaged1 = average_over_realizations(t_arr, Gamma, N, Psi_init1)
wf_sq_averaged2 = average_over_realizations(t_arr, Gamma, N, Psi_init2)

## averaged wavefunction plot
alphas_sqd1 = [np.abs(np.array(wf[0])) for wf in wf_sq_averaged1]
betas_sqd1 = [np.abs(np.array(wf[1])) for wf in wf_sq_averaged1]
alphas_sqd2 = [np.abs(np.array(wf[0])) for wf in wf_sq_averaged2]
betas_sqd2 = [np.abs(np.array(wf[1])) for wf in wf_sq_averaged2]
plt.plot(t_arr, alphas_sqd1, color='r', label='level 1 (I)', alpha=0.5)
plt.plot(t_arr, betas_sqd1, color='b', label='level 2 (I)', alpha=0.5)
plt.plot(t_arr, exponential(t_arr, Gamma, 1), color='black', alpha=0.3,
    linestyle='--')
plt.ylabel(r'sqrd amp', fontsize='xx-large')
plt.xlabel("t",fontsize='xx-large')
plt.legend(fontsize='xx-large')
plt.tight_layout()
plt.savefig('I_averaged.png')
plt.clf()

plt.plot(t_arr, alphas_sqd2, color='r', label='level 1 (II)', alpha=0.5)
plt.plot(t_arr, betas_sqd2, color='b', label='level 2 (II)', alpha=0.5)
plt.ylabel(r'sqrd amp', fontsize='xx-large')
plt.plot(t_arr, exponential(t_arr, Gamma, 0.5), color='black', alpha=0.3,
     linestyle='--')
plt.xlabel("t",fontsize='xx-large')
plt.legend(fontsize='xx-large')
plt.tight_layout()
plt.savefig('II_averaged.png')
plt.clf()


for i in [1,2,3,4,5]:
    wf_single1 = evolve_wavefunction_finite(t_arr, Gamma, Psi_init1)
    wf_single2 = evolve_wavefunction_finite(t_arr, Gamma, Psi_init2)
    ## single wavefunction plot (initial data 1)
    alphas_sqd1 = [np.abs(np.array(wf[0]))**2 for wf in wf_single1]
    betas_sqd1 = [np.abs(np.array(wf[1]))**2 for wf in wf_single1]
    plt.plot(t_arr, alphas_sqd1, color='r', label='level 1 (I)')
    plt.plot(t_arr, betas_sqd1, color='b', label='level 2 (I)')
    plt.ylabel(r'sqrd amp', fontsize='xx-large')
```

```
    plt.xlabel("t",fontsize='xx-large')
    plt.legend(fontsize='xx-large')
    plt.tight_layout()
    plt.savefig('I/'+str(i)+'I_single.png')
    plt.clf()
    ## single wavefunction plot (initial data 2)
    alphas_sqd2 = [np.abs(np.array(wf[0]))**2 for wf in wf_single2]
    betas_sqd2 = [np.abs(np.array(wf[1]))**2 for wf in wf_single2]
    plt.plot(t_arr, alphas_sqd2, color='r', linestyle='--', label='level
    1 (II)')
    plt.plot(t_arr, betas_sqd2, color='b', linestyle='--', label='level 2
     (II)')
    plt.ylabel(r'sqrd amp', fontsize='xx-large')
    plt.xlabel("t",fontsize='xx-large')
    plt.legend(fontsize='xx-large')
    plt.tight_layout()
    plt.savefig('II/'+str(i)+'II_single.png')
    plt.clf()


Unnormalized_Psi_ini0 = np.array([1,0])
Psi_init0 = normalize(Unnormalized_Psi_ini0)
wf_single0 = evolve_wavefunction_finite(t_arr, Gamma, Psi_init0)
## single wavefunction plot (initial data 1)
alphas_sqd1 = [np.abs(np.array(wf[0]))**2 for wf in wf_single0]
betas_sqd1 = [np.abs(np.array(wf[1]))**2 for wf in wf_single0]
plt.plot(t_arr, alphas_sqd1, color='r', label='level 1 (I)')
plt.plot(t_arr, betas_sqd1, color='b', label='level 2 (I)')
plt.ylabel(r'sqrd amp', fontsize='xx-large')
plt.xlabel("t",fontsize='xx-large')
plt.legend(fontsize='xx-large')
plt.tight_layout()
plt.savefig('III_single.png')
plt.clf()
```

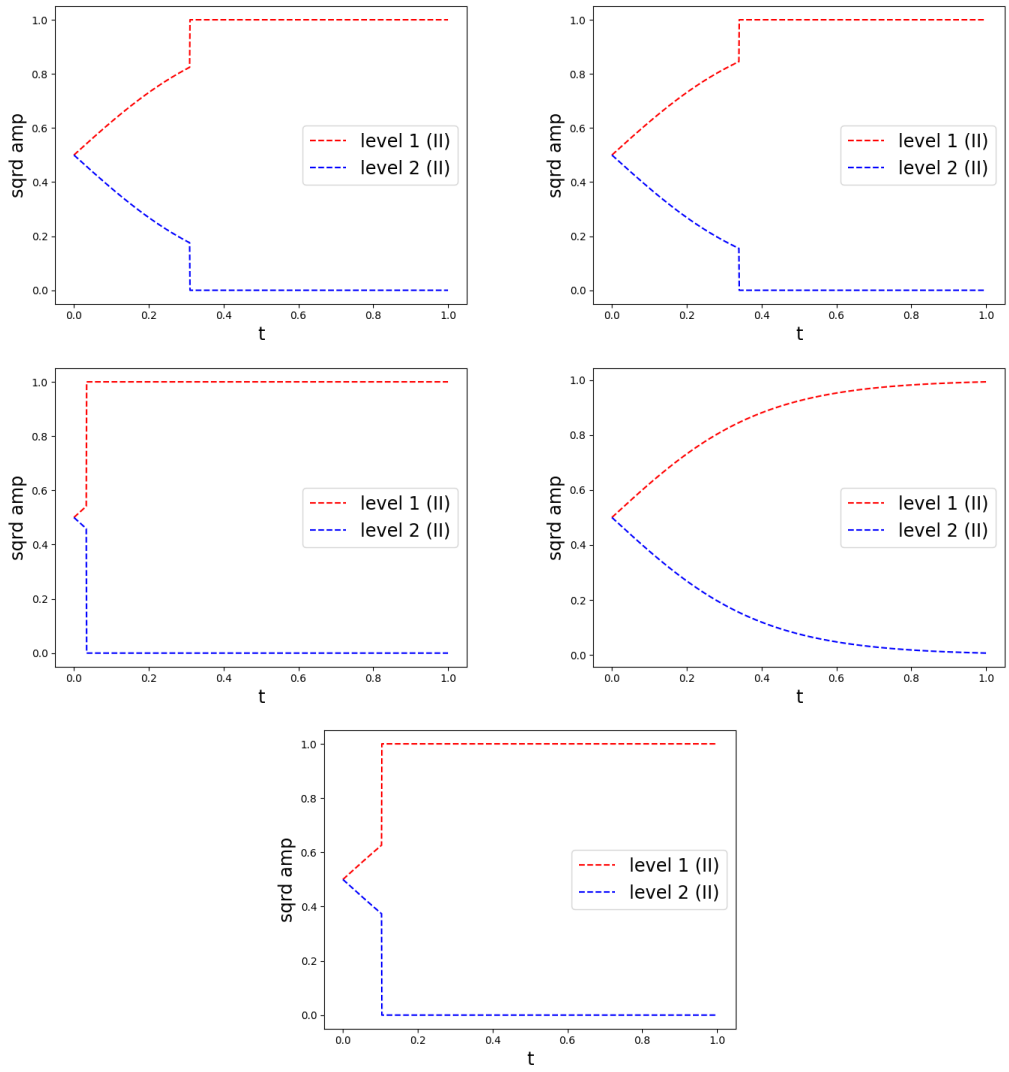
(c) For initial state $|III\rangle = |1\rangle$ being the ground state, we plot a realization with $\Gamma = 5$.



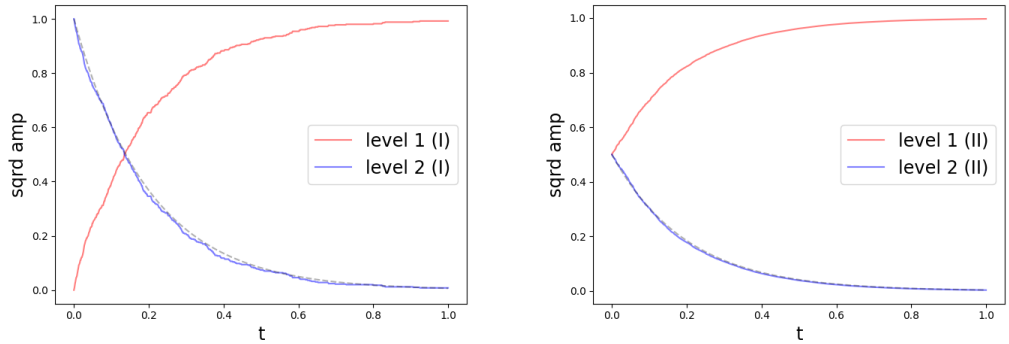
(a) For initial state $|I\rangle = |2\rangle$ being the excited state, we plot 5 realizations with $\Gamma = 5$.



(a) For initial state $|II\rangle = (|1\rangle + |2\rangle)/\sqrt{2}$, we plot 5 realizations with $\Gamma = 5$.



(b) For $|I\rangle$ and $|II\rangle$ averaged over 500 realizations, we plot the squared amplitudes (in solid lines) comparing with the exponential decay curve (in gray dashed lines).



We see here the solid curves (computed via MC method) follows closely with the dashed curves (exponential decay curve).