Instructor: Dr. S. Hughes

Due date (nominal): Sunday, Jan 23th (midnight)

Extreme Nonlinear Optics: Dynamics of Coupled ODEs

Keywords: Euler, RK4, Simple `matplotlib` Graphics, FFTs, `odeint` (from `scipy`), Optical Bloch Equation, Rotating-Wave Approximation, Beyond the Rotating-Wave Approximation, Rabi Flopping, Area Theorem.

Task: go through all the questions and coding exercises below and then write up a small physics report in the LaTeX template provided: Simple Report Template

LaTeX File of Assignment: (for convenience with your own write up) PS1

Marks: Codes (10), Report (10), Results (10). Total: (30)

Reminder: All codes must run under Python 3.x. and the Spyder IDE (check this before submitting)

Some Background:

From Rubin Landau's web page on Computational Physics (hyperlinks):
(1) Numerical Solution to Differential Equations (basic, so not needed for most of you): video (31 mins)
(2) ODE Algorithms: video (36 mins)
From Wikipedia:
Rabi cycle and Rabi flopping
Three Relevant Research Papers (all of which were published in *Physical Review Letters*), and you can probably find lots more on this general topic:
Breakdown of the Area Theorem: Carrier-Wave Rabi Flopping of Femtosecond Optical Pulses
Signatures of Carrier-Wave Rabi Flopping in GaAs
Carrier-Wave Rabi-Flopping Signatures in High-Order Harmonic Generation for Alkali Atoms

Question 1

In the class notes, we introduced the Optical Bloch equations (OBEs) in various forms. The simplest form was for CW excitation (*continuous wave*, monochromatic harmonic wave) in the RWA (rotating-wave approximation) and the interaction picture:

$$\boxed{\begin{aligned} \frac{du}{dt} &= -i\Delta_{0L}u + i\frac{\Omega_0}{2}(2n_e - 1) \quad (1)\\ \frac{dn_e}{dt} &= -\Omega_0 \mathrm{Im}[u], \quad (2)\end{aligned}}$$

where $u{=}\rho_{eg}$ is the (slowly varying) *complex* coherence and $n_e{=}\rho_{ee}$ is the population density of the excited state of a two-level system (TLS), or "quantum bit" (qubit).

(a) Begin with the example "sample bad code" onQ (ODESolver_Bad1.0.py) (deliberate!). The basic core of the code has two features:

(i) The OBEs to send to an ODE solver:

```
"ODEs - with simple CW excitation"
def derivs(y,t): # derivatives function
    dy=np.zeros((len(y)))
    #dy = [0] * len(y) # could also use lists here which can be faster if
                       # using non-vectorized ODEs "
    dy[0] = 0.
    dy[1] = Omega/2*(2.*y[2]-1.)
    dy[2] = -Omega*y[1]
    return dy
```

(ii) A *vectorized* ODE solver using the simple Euler method:

```
def EulerForward(f,y,t,h): # Vectorized forward Euler (so no need to loop)
# asarray converts to np array - so you can pass lists or numpy arrays
    k1 = h*np.asarray(f(y,t))
    y=y+k1
    return y
```

[Note: You can delete the `np.asarray` is using `numpy` arrays, which will then be very slightly faster (probably). If you usually use `numpy` arrays (and you should!), you can remove these.]

For on-resonance excitation ($\Delta_{0L} = 0$), the sample code solves for five complete Rabi oscillations in time, and compares the numerical solution with the analytical solution for the excited state population: $n_e(t) = \cos^2(\Omega_0 t)$, when $n_e(0) = 0$. In normalized time units, we use the Rabi frequency $\Omega_0 = 2\pi$, and so the final time is $tend = 5$. ($n2\pi$ gives n full cycles). The time step in the example code is 0.001 ("ridiculously small"), and as you can see the solution has still not converged. This code uses 1000 points per period, which is well within the general rule of thumb of having 100 or 1000 points over the characteristic time (or length) scale, such as the Rabi period here. This "problem" is well known in scientific computing, and is why the usual first-try approach we prefer is the Runge-Kutta fourth-order method (aka: "RK4"). Every decent physicist needs to know how to write up some RK4 code ☺, so we will get this out of the way with the exercises below.

(b) Implement an RK4 ODE solver and compare the solution with the "toy" Euler solution. Plot all three solutions (analytic, Euler, RK4), in a way that one can see (distinguish) the three solutions.

(c) Reduce the step size to the "rule of thumb" mentioned above, so h=0.01 (see also Landau's video). Now plot the RK4 solution versus the analytical solution (this problem is commonly encountered in showing similar results, and requires attention to the graphical outputs for your report, and line types).

(d) Substantially improve the graphics and potting (defaults in most languages are terrible!) to create two nice graphs that you can include in your LaTeX report (see template). Label everything clearly with fonts and lines that show all the features, lines, and satisfy the criteria of a good figure in a Physics Journal paper, such as *Physical Review B*. Keep in this style/template moving forward, so you do not have to keep working with bad figures (and so I do not have to keep reminding you and taking lots of points off for bad graphs that hurt my eyes ☹). Indeed, it is worth spending the time to create various *templates* for figures (single figure, 2 panel, 4 panel, simple animation, etc, though you can also do this when the time comes.)

Question 2

Now that you have a working ODE solver that is efficient and has been numerically checked against an analytic solution (with hopefully excellent agreement), we want to explore the OBEs with a time-dependent pulse (e.g., a pulsed laser). We will use the standard RWA equations:

$$\frac{du}{dt} = -\gamma_d u - i\Delta_{0L} u + i\frac{\tilde{\Omega}(t)}{2}(2n_e - 1) \tag{3}$$

$$\frac{dn_e}{dt} = -\tilde{\Omega}(t)\mathrm{Im}[u], \tag{4}$$

and consider a Gaussian pulse:

$$\tilde{\Omega}(t) \to \tilde{\Omega}_{\mathrm{Gauss}}(t) = \Omega_0 \exp\left(-t^2/t_p^2\right), \tag{5}$$

where t_p is the pulse width.

Initially, we consider on-resonance excitation and no dephasing (as before), so $\Delta_{0L} = 0$ and $\gamma_d = 0$. In units of normalized time $t \to t/t_p$ (say $\tilde{t}$), consider a pulse area of 2π, and confirm numerically that a perfect Rabi flop occurs, namely n_e goes from 0 to 1 and exactly back to zero at the end of the pulse. Run your simulation for a total time of $tend = 10$ (so 10 pulse durations), and add a sensible offset to the pulse (e.g., $5t_p$, or 5 in normalized units, is appropriate, ensuring that the pulse is turned on well away from the center of the pulse, so it starts from almost zero.)

Question 3

(a) Now add in some finite laser detuning from the TLS resonance, Δ_{0L}, and vary it from zero to Ω_0. Make a graph of peak n_e (maximum) versus detuning.

(b) Do the same thing for a finite dephasing, while setting the detuning to zero again.

Question 4

Next we will solve the full Rabi problem with no rotating wave approximation (exciting stuff!). We are going into a domain where no analytical solution exists. We will consider the full-wave Bloch equations:

$$\boxed{\begin{aligned}\frac{du}{dt} &= -\gamma_d u - i\omega_0 u + i\Omega(t)(2n_e - 1) \quad (6)\\ \frac{dn_e}{dt} &= -2\Omega(t)\mathrm{Im}[u], \quad (7)\end{aligned}}$$

where u is now quickly-varying (you may have to adjust your time step appropriately), and $\Omega(t)$ is a full-wave Rabi field:

$$\Omega(t) \to \Omega_{\mathrm{Gauss}}(t) = \Omega_0 \exp\left(-t^2/t_p^2\right) \sin(\omega_L t + \phi), \quad (8)$$

and initially choose $\phi = 0$ and $\gamma_d = 0$. We will stay on resonance, so $\omega_0 = \omega_L$.

(a) Using a nominal 2π pulse (as defined from the RWA envelope solution), investigate the features of $u(t)$ and $n_e(t)$ when $\omega_L = 20\Omega_0, 10\Omega_0, 5\Omega_0, 2\Omega_0$. In normalized time units, with $t_p = 1$, then a 2π pulse is obtained when $\Omega_0 = 2\sqrt{\pi}$, and you should derive this explicitly in your write up (namely, derive the condition for a 2π pulse). The general solution is well known and called the "Area Theorem." Show the numerical results graphically and compare and contrast with the RWA solution. For the latter example of $\omega_L = 2\Omega_0$, also show the solution using $\phi = \pi/2$, and comment on your findings. Can the RWA models account for the phase of the drive, and would it make any difference to the solution?

(b) For the example laser frequency of of $\omega_L = 4\sqrt{\pi}$, investigate what happens when the pulse area changes from 2π to 10π and 20π. Comment on your findings.

(c) In ultrashort pulse experiments, it is common to measure the frequency content of the transmitted or reflected pulse, through the power spectrum, $|E(\omega)|$. In a thin-sample approximation (the medium that contains the TLS material), we can relate to this quantity by studying the power spectrum of the polarization, which is related to the coherence of the OBEs: $P(t) \propto \mathrm{Re}[u(t)]$, from which you can obtain $|P(\omega)|$.

Using a polarization decay rate of $0.4/t_p$, plot the polarization power spectrum in normalized units of ω/ω_L for $|P(\omega)|$ for the three cases studies in (b) - fixed a small typo here. You will also need to increase your simulation time to ensure that the transient has decayed to approximately zero (such as $50t_p$). You can use the `numpy` functions for your FFTs (fast Fourier transforms), including, e.g., `np.fft.fft`, `np.fft.fftfreq`, and `np.fft.fftshift`. You should become familiar with these functions as they are typically used quite often. In sensible normalized units, plot the solution on a y log scale (e.g., using `semilogy`), and only show ω/ω_L ranging from 0 to 6 (so 6 times the laser frequency). Comment on the features you obtain.

(d) Finally, using the final example above, compare your code results and run times (give the results from a python timer, e.g., using `import timeit`, `start = timeit.default_timer()`, `stop = timeit.default_timer()`) versus the python `odeint` routine available from `SciPy`. You can also try the Matlab-style tictoc if you like.

Some pseudo-code for `odeint` is given below:

```
# read in ode solver
from scipy.integrate import odeint
...
y0=[0.,0.,0.] # initial condition
# options can also be includes here for errors and additional step sizes, such
                                          as mxstep=20 would allow up to 20
                                         sub divisions of your chosen time
                                         array step
y=odeint(OBEsFull,y0,t)
pop = y[:,2]
```

Note `odeint` cannot deal with complex numbers (not a big deal for our fairly simple examples in this assignment, as we can work out the real and imaginary components by hand), and you can easily separate u into real and imag parts. However, odeintw can, but we do not need to look into that right now (but feel free to also try out that solver if you want to use complex arrays). You need to download `odeintw` and then can include at the start of your code:

```
from odeintw import odeintw
```

Using this routine is very similar to `odeint`, but you can now use complex arrays if you wish (with many vector arrays such as a matrix of ODEs with complex parts, this is a much cleaner way to solve the problem). With your own RK4, it is also easy to use complex arrays if you wish, and you can declare these in `numpy` (e.g., `C = zeros((2,2),dtype=complex)`).

Note also than alternatively you could test using one of the `solve_ivp` routines, which are newer and also from `scipy`, but they have a slightly different syntax and tend to be a bit slower (extra overhead from trying to mimic Matlab-style ODE suites), but feel free to use one of those instead if you are more familiar with it, or two both; the default method for the solver is RK45, which you can also call explicitly by passing: `method='RK45'`). In these `scipy` routines, you can also explicitly give the absolute and relative errors, which can have certain advantages (as they will usually adjust the step size accordingly – "adaptive step", which is beyond what we will do just now). These can also be used to benchmark the accuracy of your own solvers, though we can always write an adaptive step to do that as well. We come back to these ODE solvers for later assignments, and you will probably be using your RK4 for the rest of your life, so add some sensible comments so you (and others) know whet is going on.