Problem Set 3

Submit your solution via NTULearn, with one Python source file per problem. Problems labeled like this—(5* marks)—are optional for undergraduates, and can be attempted for a bonus of 1/5 the indicated marks; for graduate students, the problems are compulsory and receive full marks. Follow good programming style, and label output plots clearly.

0. MAGNETIC FIELDS PRODUCED BY CURRENT COILS

Consider a circular current loop, as shown in Fig. 1(a). The current is I, and the loop has radius R; the wire is infinitesimally thin. For convenience, let the coordinate origin be at the center of the loop, with the loop lying in the x-y plane as shown in Fig. 1(a). According to the Biot-Savart law, the magnetic field at $\vec{r} = [x, y, z]$ is

$$\vec{B} = \frac{\mu_0 I R}{4\pi} \int_0^{2\pi} d\varphi \, \frac{z \sin(\varphi)\, \hat{\rho} + [R - \rho \sin(\varphi)]\, \hat{z}}{\left[R^2 + \rho^2 + z^2 - 2\rho R \sin(\varphi)\right]^{3/2}}, \tag{0}$$

where μ_0 is the permeability of free space, $\rho \equiv \sqrt{x^2 + y^2}$ is the radial distance in cylindrical coordinates, and $\hat{\rho}$ is the unit vector pointing in the cylindrical radial direction (i.e., pointing from the z axis to $\vec{r}$).

(a) (2 marks) Write the following function to compute the field produced by a circular

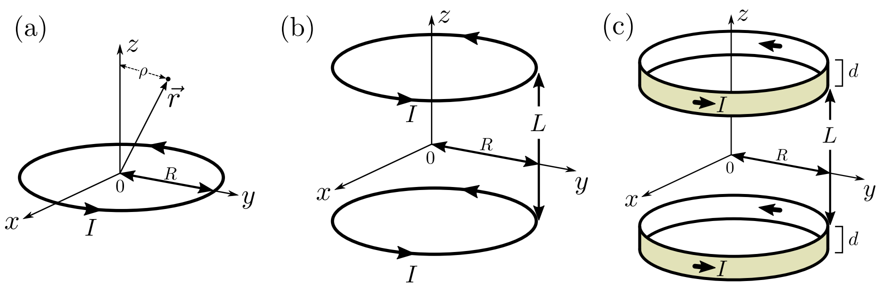

FIG. 1: (a) A single loop of radius R, carrying current I, centered at the origin and parallel to the x-y plane. (b) A Helmholtz coil, consisting of two loops each carrying current I, with radius R and separated by L. We put the coordinate origin at the mid-point between the loops. (c) A modified Helmholtz coil where each loop has finite length d along the z direction.

current loop, at a specified position $\vec{r}$ relative to the center of the loop:

`def coil_field(position, current, radius):`	
Inputs	
`position`	A 1D array $[x, y, z]$, specifying a position $\vec{r}$ in Cartesian coordinates relative to the center of the loop. (The loop lies in the x-y plane.)
`current`	The current I.
`radius`	The loop radius R.
Output	
`B`	The magnetic field at position $\vec{r}$, in Cartesian coordinates.

Take $\mu_0 = 1$ for simplicity. Use `scipy.integrate.quad` to perform the integral.

(b) (3 marks) A "Helmholtz coil" is commonly used in physics experiments to produce a highly uniform magnetic field. It consists of two parallel circular loops of radius R, separated by distance L, with each loop carrying equal current I. Assume the origin is exactly between the two loops, as shown in Fig. 1(b). Write the following function:

`def coil_demo(z, rmax, current, radius, spacing):`	
Inputs	
`z`	The z coordinate at which to plot the magnetic fields (see below).
`rmax`	The maximum value of the radial coordinate ρ for plotting (see below).
`current`	The coil current I.
`radius`	The coil radius R.
`spacing`	The inter-coil separation L.

This function has no return value. It should create two subplots: (i) B_z (the z component of the magnetic field) versus ρ (the radial cylindrical coordinate), and (ii) B_ρ (the ρ component of the magnetic field) versus ρ. The plots should extend from $\rho = 0$ to `rmax`; you are free to choose the discretization.

Discuss in code comments: Qualitatively, describe how the magnetic field behaves for $z = 0$ and $R = L$. What about different values of z? What is the behavior for $R > L$ or $R < L$?

(c) (4* marks) Suppose we have the situation shown in Fig. 1(c), where each coil has finite length d extending in the z direction. (The thickness in the ρ direction remains negligible.) The current in each coil is I, flowing uniformly on the coil area. Write the following function:

```
def coil_demo_2(z, rmax, current, radius, spacing, length):
```

Inputs	
`z`	The z coordinate at which to plot the magnetic fields.
`rmax`	The maximum value of the radial coordinate ρ for plotting.
`current`	The coil current I.
`radius`	The coil radius R.
`spacing`	The inter-coil separation L.
`length`	The length d of each coil.

Similar to part (b), this should plot (i) B_z versus ρ and (ii) B_ρ versus ρ.

1. STIFF EQUATIONS

The Van der Pol equation is the following ordinary differential equation:

$$\frac{d^2x}{dt^2} - \mu(1-x^2)\frac{dx}{dt} + x = 0. \tag{1}$$

It occurs when describing relaxation oscillations in certain nonlinear electrical circuits.

(a) (3 marks) Write a function `vanderpol_plot(x0, v0, t1=500., nt=2000, mu=100.)`, which plots $x(t)$ versus t for `nt` discrete times in the range $0 \le t \le$ `t1`, with initial conditions $x(0) =$ `x0` and $\dot{x}(0) =$ `v0`. The input `mu` specifies the μ parameter in Eq. (1). This function should use Scipy's ODE class `scipy.integrate.ode`, with any integration scheme you like.

(b) (3 marks) During each single run of the integrator (i.e., one invocation of the `integrate(t)` method), the ODE's derivative function gets called a certain number of times. The number of calls is decided by the integrator, based on the time step Δt and other factors.

Write a function `vanderpol_profile(dt1=1e-3, dt2=10., mu=100.)`, which integrates Eq. (1) over time Δt, using one `integrate(t)` call; the interval Δt should range from `dt1` to `dt2`. The function should plot the number of calls to the ODE's derivative function, versus Δt. In the same plot, show results for DOPRI5 (a Runge-Kutta solver), and for LSODA (an implicit solver). *Note*: you may need to increase the `nsteps` parameter for DOPRI5.

2. PENDULUM MOTION

The equation of motion of a harmonically driven pendulum is

$$\ddot{\theta} + 2\gamma\dot{\theta} + \omega_0^2 \sin\theta = F_d \cos(2\pi t), \tag{2}$$

where θ is the pendulum angle, γ is the damping constant, ω_0^2 is the pendulum constant, and F_d is the magnitude of the driving force. The drive frequency is normalized to 2π.

(a) (3 marks) Write a function to solve the pendulum ODE:

`def pendulum_sample(theta0, thetadot0, gamma, omega0, Fd, t):`	
Inputs	
`theta0`	Initial condition $\theta(t_0)$ (a number).
`thetadot0`	Initial condition $\dot{\theta}(t_0)$ (a number).
`gamma`, `omega0`, `Fd`	The pendulum parameters γ, ω_0, and F_d, as defined in Eq. (2).
`t`	A 1D array of times at which to report the solution. The first element in this array is the initial time t_0 for the initial conditions.
Return values	
`theta`	A 1D array containing the values of $\theta(t) \in [0, 2\pi]$ at the times in `t`.
`thetadot`	A 1D array containing the values of $\dot{\theta}(t)$ at the times in `t`.

Note that the output $\theta(t)$ should be given modulo 2π.

(b) (3 marks) Write a function `pendulum_demo()`, which plots the angular velocity $\dot{\theta}(t)$ vs t. (Note: it's more convenient to plot $\dot{\theta}$ instead of θ, since θ is defined modulo 2π.) Use the parameters $\gamma = 0.1$ and $\omega_0 = 10$, and in separate subplots show the results for $F_d \in \{40, 50, 60, 70, 80\}$. You are free to choose appropriate initial conditions at $t = 0$ and maximum t.

(c) (3 marks) To understand the results of part (b), we will perform a Fourier analysis of $\dot{\theta}(t)$. Write a function `pendulum_spectrum_demo()`, which plots the Fourier power spectrum of $\dot{\theta}$, using the same parameters as in (b). Show the results for each value of F_d in a separate subplot. Ensure that the horizontal (frequency) axis has the right numerical factor corresponding to the Fourier transform; you may also want to adjust the axis limits so that the key features of the spectrum are clearly visible. The vertical axis can be scaled arbitrarily. You should use FFTs to compute the Fourier spectra.

(e) (3* marks) Write a function `pendulum_perturbation_demo()` to test the sensitivity of the numerical results to perturbations in the initial conditions. In this function, plot $|\Delta\dot{\theta}(T)|$ versus $F_d \in [50, 55]$, where $\Delta\dot{\theta}(T)$ denotes the deviation in the angular velocity at the final time T, when the initial angle is increased slightly from $\theta(0) = 1$ to $\theta(0) = 1 + 10^{-8}$. Take $\dot{\theta}(0) = 0$, $T = 50$, and take all other oscillator parameters to be the same as before.

Discuss in code comments: What do you observe, and what are the implications for the accuracy of numerical ODE solvers?

(f) (3* marks) Write a function `pendulum_turning_points_demo()`, which displays a scatter plot of $\theta|_{\dot{\theta}=0}$ versus F_d. Here, $\theta|_{\dot{\theta}=0}$ refers to the *turning points* of the pendulum motion: the values of θ whenever $\dot{\theta} = 0$. Choose a horizontal range $50 < F_d < 80$, with all other pendulum parameters the same as in parts (b)–(d).

When calculating $\theta|_{\dot{\theta}=0}$, you should "warm up" the simulation by running it for sufficiently many time steps before hunting for the turning points, in order to remove the effects of the transients. It is up to you to devise a suitable way to locate the turning points.
Discuss in code comments: How are these results related to the results of part (d)?