P3800 Project 2: Ordinary Differential Equations

1 Introduction

For this project you will be asked to numerically solve the equations of motion for a charged particle in magnetic and electric fields. This physics problem is described starting on page 10 of the MacKinnon notes (see course website), and we reproduce the relevant parts here.

Consider a positively charged particle moving in the presence of constant electric and magnetic fields. In the case where the particle experiences a drag force, the equation of motion in vector notation is,

$$m\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = q\mathbf{v}\times\mathbf{B} - \gamma\mathbf{v} + q\mathbf{E}. \tag{1}$$

In our case, the particle is initially at the origin, $\mathbf{B} = (0, 0, B)$, $\mathbf{E} = (0, E, 0)$, and the initial velocity $\mathbf{v_0} = (0, v_0, 0)$, and hence the motion of the particle is confined to the xy plane.

As discussed in the MacKinnon notes, it is appropriate to introduce a new dimensionless time variable $\tau = t\ qB/m$. Further, it is convenient to also introduce new position variables $\tilde{x} = x\ qB/m$ and $\tilde{y} = y\ qB/m$. After making the transformations, Eq. 1 in component form becomes,

$$\begin{aligned}
\frac{\mathrm{d}v_x}{\mathrm{d}\tau} &= +v_y - \frac{\gamma}{qB}v_x, &(2)\\
\frac{\mathrm{d}v_y}{\mathrm{d}\tau} &= -v_x - \frac{\gamma}{qB}v_y + \frac{E}{B}, &(3)\\
\frac{\mathrm{d}\tilde{x}}{\mathrm{d}\tau} &= v_x, &(4)\\
\frac{\mathrm{d}\tilde{y}}{\mathrm{d}\tau} &= v_y. &(5)
\end{aligned}$$

The parameters of the system have values (in SI units) $B = 10$ T, $q = 1.6\times 10^{-16}$ C, $m = 1.6\times 10^{-21}$ kg, $\gamma = 8.0\times 10^{-17}$ kg s^{-1}, $E = 10$ N C^{-1}, $v_0 = 10$ m s^{-1}.

How does $\tilde{x}(\tau)$ relate to $x(t)$? The factor $qB/m = 10^6$ s^{-1}, so going from $\tau = 0$ to $\tau = 1$ corresponds to a real time difference of 10^{-6} s, or 1 μs. Furthermore, if $\tilde{x} = 1$ m s^{-1}, then $x = 10^{-6}$ m or 1 μm (yes, $\tilde{x}$ has dimensions of velocity), but $\frac{\mathrm{d}\tilde{x}}{\mathrm{d}\tau} = 1$ is just 1 m s^{-1}. When the dust settles, solving Eqs. 2-5 yields $\tilde{x}(\tau)$, the graph of which is identical to $x(t)$ when x is in μm and t in μs. Fig. 1 shows

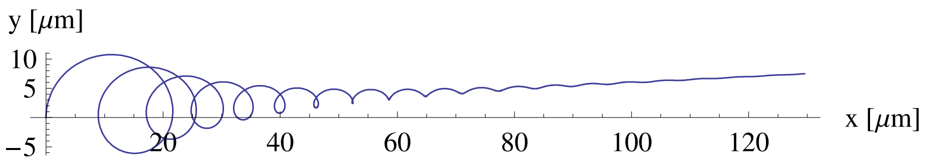

Figure 1: Trajectory of charged particle undergoing motion in constant crossed electric and magnetic fields with drag as described in the Introduction.

the particle trajectory in the xy plane for the first 120 μs. The plot was actually obtained by plotting $\tilde{y}(\tau)$ and $\tilde{x}(\tau)$ parametrically for τ from 0 to 120, and then relabeling the axes. For small drag, the dominant direction of the particle's trajectory is in the direction of $\mathbf{E} \times \mathbf{B}$, not in the direction of the electric field. For long times, the slope of the trajectory $\frac{\mathrm{d}y}{\mathrm{d}x}$ is proportional to γ.

This problem can be solved analytically. The value of x at $t = 120$ μs is 129.554231971262952559848 μm.

2 Euler

For this section, you are required to write a program that uses Euler's method to numerically integrate the transformed equations of motion Eqs. 2-5. To do this, it is helpful to rewrite Eqs. 2-5 in terms of a vector of dependent variables $(y_1, y_2, y_3, y_4) = (\tilde{x}, \tilde{y}, v_x, v_y)$ to get,

$$\frac{\mathrm{d}y_1}{\mathrm{d}\tau} = y_3, \tag{6}$$

$$\frac{\mathrm{d}y_2}{\mathrm{d}\tau} = y_4, \tag{7}$$

$$\frac{\mathrm{d}y_3}{\mathrm{d}\tau} = +y_4 - \frac{\gamma}{qB} y_3, \tag{8}$$

$$\frac{\mathrm{d}y_4}{\mathrm{d}\tau} = -y_3 - \frac{\gamma}{qB} y_4 + \frac{E}{B}. \tag{9}$$

Your code should include a function or subroutine that returns the values of $\frac{\mathrm{d}y_i}{\mathrm{d}\tau}$, given $\mathbf{y}$. This function, in turn, should be utilized by a subroutine that iterates Euler's scheme, i.e. carries out,

$$y_i(\tau + \Delta\tau) = y_i(\tau) + \frac{\mathrm{d}y_i(\mathbf{y}(\tau), \tau)}{\mathrm{d}\tau} \, \Delta\tau.$$

Finally, there should be a main program that controls the whole process, provides output, etc.

Plot the trajectory of the particle for 120 μs, using a timestep of $\Delta\tau = 0.001$. Report the position of the particle at $t = 120$ μs and the relative error in the x-coordinate. Include your code in your submission.

3 Euler with Adaptive Timestep

In your class notes on Euler adaptive timestep methods, there is an example where an estimate of the truncation error is compared to a desired tolerance. The timestep is accordingly multiplied or divided by 2 in order to find an approximate value of the largest $\Delta\tau$ that will satisfy the desired tolerance. Come up with a scheme of your own that uses a predefined tolerance to change the timestep. For example, you could use the truncation error estimate itself to determine how the stepsize is reduced or increased.

In your report, include a description of your prescription for varying $\Delta\tau$. Plot the trajectory $y(x)$, the x-coordinate $x(t)$ and the timestep $\Delta\tau(\tau)$ after selecting an appropriate tolerance. Include your code in your submission. Discuss your results in terms of steps taken, accuracy of result and anything you wished to explore, like varying the tolerance.

4 RK45

In the class notes there is a discussion of an adaptive timestep method utilizing Runge-Kutta fourth and fifth order methods simultaneously to estimate the error. The relevant sections and source code from *Numerical Recipes* are included in relevant subdirectories in the project package. There is source code in Fortran90 and C. The main codes are `odeint`, `rkck` and `rkqs`. There are also additional source code files for C and Fortran90 that are necessary for making use of the main routines.

You are required to write a routine `derivs` that returns the values of the derivatives of the dependent variables, as well as a driver routine that calls `odeint`. Use the code in the language of your choice to solve the system of equations 6-9 and then plot the trajectory from $\tau = 0$ to $\tau = 120$. Also, plot $\Delta\tau(\tau)$ (you will need to modify `odeint` to do this). Report the final position of the particle at 120 μs, and the relative error in the x-coordinate. In carrying out the above, use the following parameters required by `odeint`: required accuracy `eps` $= 1 \times 10^{-6}$, initial step size `h1` $= 1 \times 10^{-5}$ and trajectory-saving increment `dxsav` $= 1 \times 10^{-5}$. How many points along the trajectory does the routine actually calculate? Briefly discuss these results in comparison to the simple Euler and adaptive stepsize Euler results.

As a start, there is in the project package an example (in both C and Fortran90) in which the simple harmonic oscillator is solved using *Numerical Recipes* routines `rk4` and `rkdumb`, i.e., the ones for fixed timestep fourth-order Runge-Kutta.

5 Your Submission

Your written report should answer the specific questions posed in Sections 2-4, and include relevant graphs with captions, axis labels etc. Include in your submission all source code, Makefiles, and a script called run_project2.sh that compiles all code and produces a summary of results. Your makefile should include rules for generating figures from the data. As usual, code for generating data should be written in C or Fortran. Feel free to use code from lecture on Euler and the midpoint scheme as a starting basis for your code. Please use Python for generating graphs for this project.