

PHYS 218 – S07 –Assignment #2 – Oscillations and Geometry

In this assignment, we will look at some systems that oscillate, sometimes regularly, but sometimes *chaotically*. In addition, we will see how geometric ideas can help us analyze these oscillations. Along the way, we will test the performance of three integration methods.

1) A prototypical oscillator is the simple pendulum (a mass with mass m hanging from a string (here assumed rigid if the pendulum does a “loop-the-loop”) of length l). You can describe the motion of the pendulum with two coupled ODEs:

$$\begin{aligned}\frac{d\omega}{dt} &= -\frac{g}{l}\sin(\theta) \\ \frac{d\theta}{dt} &= \omega\end{aligned}$$

where θ is the angle the pendulum mass makes with the vertical and ω is the angular velocity (the time rate of change of θ). If we chose to measure time, not in seconds, but in units of $t' = \sqrt{l/g}$ and angular velocity in units of $\omega' = \sqrt{g/l}$, then you can show that the ODEs become completely dimensionless:

$$\begin{aligned}\frac{d\bar{\omega}}{d\tau} &= -\sin(\theta) \\ \frac{d\theta}{d\tau} &= \bar{\omega}\end{aligned}$$

where $\tau = t/t'$ and $\bar{\omega} = \omega/\omega'$.

Product 2.1: Write three computer programs in C (or C++) that integrates the dimensionless equations just above using a) the Euler method, b) the second-order Runge-Kutta method, and c) the fourth-order Runge-Kutta method. Feel free to reuse parts of the programs you wrote for assignment #1. Write your programs so that it is easy to adjust the time step and total length of time to simulate.

We want to see how to check that our simulations are accurate. In class, I mentioned three possibilities: a) internal consistency (*e.g.*, does the simulation change significantly if I reduce the time step?), b) compare with theory where you can, and c) check that conservation laws are obeyed.

One can show that for small values of $\theta_0 \ll 1$ and $\omega_0 = 0$, theory predicts that the period of the oscillation will be 2π . As this is a mechanical system without any friction, we should also conserve energy at every time step. In these dimensionless units, the (dimensionless energy) is given by:

$$\varepsilon = \frac{\bar{\omega}^2}{2} + (1 - \cos \theta).$$

For each of the three methods, run simulations with initial conditions $\bar{\omega}_0 = 0$ and $\theta_0 = 0.1$ for a total time of 50 (in dimensionless time). For each of the three methods run the simulations with a time step of 0.5, 0.1, and 0.05. (Thus there will be nine total runs.) For each run, output a file containing τ , θ , $\bar{\omega}$, and ε at each time step. For each run, plot θ vs. τ and ε vs. τ . Calculate the oscillation period for each of the runs. Comment on your results in light of a) internal consistency, b) comparison with theory, and c) conservation of energy.

2) Now we want to explore the geometry of the pendulum's phase space by creating a phase space plot.

Product 2.2: Use the most precise method and time step from part 1 (but only print out results every 0.2 time units). Create a phase space plot with θ on the horizontal axis and ω on the vertical axis. Make your limits $-2\pi \leq \theta \leq 2\pi$ and $-3 \leq \bar{\omega} \leq 3$. Do 20 runs all with the initial value of $\theta = 0$ and the initial value of $\bar{\omega}_0 = (\sqrt{2}/10) \times n$, with $n = 1, 2, \dots, 20$. Note: in some of your runs, the code will produce values of θ outside the given plot range. Add or subtract a multiple of 2π until the points are within the plot range. Identify all stable and unstable fixed points on your plot. Show an example of a periodic and a non-periodic orbit.

3) We are now starting a transition to using the hard work of others to ease our own work. In the class folder on P:\Class you will find a subfolder \Assignment#2\NumRecRK4. This folder contains a) a C function that implements a single fourth order Runge Kutta time step (rk4.c), b) a set of header files and utility functions that are needed for this function (nr.h, nrutil.h, nrutil.c), and c) a program that demonstrates the use of rk4 (xrk4.c). I also handed out some relevant pages from *Numerical Recipes in C* (from which all of this code is taken without permission) that describe the use of the rk4 routine. I couldn't Xerox the whole book, of course, so you should also know that for various reasons, this routine starts array indexes at 1 instead of 0 and in some cases you may need to just blindly follow the example to get things to work (or ask questions).

Product 2.3: Rewrite your code of part 2.1 to use the *Numerical Recipes in C* Runge-Kutta stepper. Using this new code, redo the runs at different time steps from Product 2.1 (just the ones involving your fourth-order Runge Kutta routine) and comment on whether the results from your fourth-order Runge Kutta routine and the *Numerical Recipes* fourth-order Runge Kutta routine are essentially the same or not. Compare them based on the three criteria a) internal consistency, b) comparison with theory, and c) conservation of energy.

To be continued....