480/905: Session 7

*Online listings:* eigen\_basis\_class.cpp and diffeq\_oscillations.cpp printouts.

Now that we've got routines to solve differential equations, we're going to explore some interesting ones: nonlinear oscillators. Today we'll play with a program that solves for the time dependence of such an oscillator.  
  
*Your goals for today (and ...):*

* If you didn't complete it, do the plot from Session 6 of relative error at t=1 vs. mesh size h.
* Think about how to enhance the eigen\_basis code with more C++ classes.
* Run a code that solves the differential equation for a (driven) nonlinear oscillator and explore how the time dependence changes as various input parameters change.
* Add friction (damping) to the code.

(Possibly) Leftover Task from Session 6

Spend no more than about 45 minutes on this.

1. **Integrating a First-Order Differential Equation.** Try to finish through part 7. If you find that the errors for Euler and Runge-Kutta lie on top of each other, most likely you have not evaluated the exact answer at precisely the same time as the last points. If you get stuck, ask an instructor.

More on C++ Classes: eigen\_basis\_class

The code eigen\_basis\_class.cpp is a simple modification of eigen\_basis.cpp to use the Hamiltonian class we introduced for eigen\_tridiagonal\_class.cpp. Here we'll take a few minutes to think about how to introduce additional classes.

1. Take a look at the eigen\_basis\_class.cpp printout and note how the Hamiltonian class is re-used without modification. (If you haven't done so yet, read the discussion of this class in the Session 7 notes.) The only tricky change is that matrix indices go from 1 to dimension rather than from 0 to dimension-1. *What parts of the Hamiltonian class implementation do you not yet understand?*

**I’m not entirely sure on the use of tht hij\_parameters pointer and the use of the switch statement**

1. The potential is another good candidate for a class. We'd like to just evaluate the potential at r without having to use constructions like the switch statement in the Hij\_integrand function with all the messy void parameters. (Think about how awkward and prone to error it is to add another potential.) *What would you like the declaration statement for the Potential class to look like? What method(s) would you like the class to have?*

**I would declare it to be a double and honestly, I’m not quite sure on what other methods I would like to add to the class.**

1. *Give at least one example of an additional class that would be useful to define.*

**A class that would plot the potentials**

Driven Nonlinear Oscillations

The Session 7 notes describe the driven nonlinear oscillator that is coded in diffeq\_oscillations.cpp. Note that the force depends on k and an exponent p, the external force has a magnitude f\_ext, a frequency w\_ext, and a phase phi\_ext. The initial conditions in position and velocity are designated x0 and v0. You also have control over the time interval (increase t\_end to see longer times), the step size h, and how often points are printed to the file (plot\_skip).

1. Use make\_diffeq\_oscillations to create diffeq\_oscillations. This code outputs to the file diffeq\_oscillations.dat five columns of data: t, x(t), v(t), kinetic energy, and potential energy. There are four gnuplot plot files provided (diffeq\_oscillations1.plt, etc.), each of which generates a different type of plot. Run diffeq\_oscillations with the default values (enter "0" when it says "What do you want to change?") to calculate a data set. Start gnuplot and "load diffeq\_oscillations1.plt" and then "load diffeq\_oscillations2.plt". (Once you've given these commands once, you can use just use the arrows to go back and forth.) *Briefly, what do each of these plots show?*

**The first plot shows the behavior of the wave function y(t) along t, which resembles a a sin graph. The second plot shows the phase space over time.**

1. Wouldn't it be convenient to generate all four plots at once in separate files? Load "diffeq\_oscillations\_all.plt"!
2. It's always a question whether or not you have coded a problem correctly, so you should always seek ways to check your results. One possibility is if we have a known solution. This works for p=2 (simple harmonic oscillator). What about other p? Another check is to identify a quantity that shouldn't change with time. *Create a plot of such a quantity (you'll want to increase t\_end) and observe the effect of changing the step size h to a larger value [e.g., try 10 and 100 times larger]. How do you decide on a reasonable h to use?*

**I decided that the most reasonable h for this case is 0.001 because the .dat file has relatively small value so having a small step can capture enough points to plot the graph. Also when the graphs are plotted for h=0.001, the graph look the best.**   
  
  
  
  
  
(The "plot\_skip" parameter indicates how often a point is written to the output file. So plot\_skip=10 means that every 10 points is output.)

1. Verify that different amplitudes (e.g., different initial conditions determined by x0 and v0) lead to different periods for an anharmonic oscillator (p<2 or p>2). [Hint: You might find the "append" option useful.] *Can you identify a qualitative rule? E.g., does larger amplitude mean shorter or longer period always? Try to explain the rule?*

**Larger amplitude always leads to shorter period because the amplitude stretches the y-axis, leading to the shrinking of the x-axis**

1. Go back to the original parameters (quit the program and start it again), which has p=2. *Now add a driving force f\_ext=10 with w\_ext=1 and look at the time dependence and phase-space plots. Then increase w\_ext to 3.14 and then to w\_ext=6.28. What are you observing? Now repeat with p=3 (starting with f=0). Can you find resonant behavior?*

**I was able to find resonant behavior when w\_ext i= 6.28 for both p =2 and p=3. This was evident in their respective time-dependence plots that resembles graphs that depicts resonant behavior**

Adding Damping

Real-world systems have friction, which means the motion will be damped. The Session 7 notes have a list of three simple models for friction. We'll implement viscous damping: Ff = -b\*v, where v(t) is the velocity.

1. *Introduce the damping parameter "b" into the code:*
   1. add it to the force\_parameters structure (with a comment!);
   2. add it to the list of local force parameters in the main program;
   3. give it an initial value;
   4. add a menu item (e.g., [13]) and a case statement to get a new value.

Try this part out before proceeding. *Did it work?*

**Yes**

1. Modify the "rhs" routine to include damping (you're on your own here!). *What did you add?*
2. double
3. rhs (double t, double y[], int i, void \*params\_ptr)
4. {
5. double x = y[0];              // local x value
6. double k = ((force\_parameters \*) params\_ptr)->k;   // local force parameters
7. double m = ((force\_parameters \*) params\_ptr)->m;
8. double p = ((force\_parameters \*) params\_ptr)->p;
9. double f\_ext = ((force\_parameters \*) params\_ptr)->f\_ext;
10. double omega\_ext = ((force\_parameters \*) params\_ptr)->omega\_ext;
11. double phi\_ext = ((force\_parameters \*) params\_ptr)->phi\_ext;
12. double b = ((force\_parameters \*) params\_ptr)->b;
13. double F\_ext = f\_ext \* cos (omega\_ext \* t + phi\_ext);
14. //double B = (-b\*y[1]) - F\_ext;
16. if (i == 0)  // first equation
17. {
18. return (y[1]);
19. }
20. if (i == 1)  // second equation
21. {
22. if (x == 0)
23. {
24. return ((F\_ext-b\*y[1]) / m); //indicates acceleration
25. }
26. else if (x < 0)
27. {
28. return ((F\_ext-b\*y[1]+ k \* pow (fabs (x), (p - 1))) / m);
29. }
30. else if (x > 0)
31. {
32. return ((F\_ext-b\*y[1] - k \* pow (fabs (x), (p - 1))) / m);
33. }
34. }
35. return (1);           // something's wrong if we get here
36. }

1. Test your routine starting with p=2 and a small damping and look at both the time dependence and the phase-space plots. Then try some other p values.

**The plots for this part is attached in the github repository**

1. *Identify the three regimes described in the Session 7 notes: underdamped, critically damped, and overdamped.*

**The plots for this part is attached in the github repository**

EXTRA: Looking for Chaos (Part I) [we'll explore in more detail in Session 8]

Now we want to put it all together: a damped, driven, nonlinear oscillator. A different system with the same basic features is the realistic pendulum, which is described in the Session 7 notes.

1. In the notes there is a list of characteristic structures that can be found in phase space, with sample pictures. *Can you find combinations of parameters that produce pictures like these?* (Try to imitate the x(t) vs. t pictures first.)