Session_07 activities worksheet

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_06 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.

A) 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_07 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 understand most of the implementation itself. The only part of the Hamiltonian class I don't really understand are the GSL-specific components. For example, I understand what gs_eigen_symmv actually does, but I don't understand how it does it at all, or the specific mechanics of some of the GSL functions.

2. 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 (the one where the Potential object is declared in the main program) for the Potential class to look like? What method(s) would you like the class to have?*

   → I would like the Potential class to be instantiated via a parameters struct, like that of

potl_params. This would help consider the different parameters different potentials need. Then you would want methods that help calculate values of the function at different x (or r, or whatever variable you need), and then you could call just a generic function on a generic Potential instance.

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

   → A specific HarmonicOscillator class could be nice, and you can maybe make it generalized based on different parameters like damping or other different parameters (like amplitude or phase shift).

B) Driven Nonlinear Oscillations

The Session_07 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 file shows the actual oscillation motion x(t) vs. t. The second shows the Hamiltonian phase space plot using v(t) vs. x(t) to get the rough phase space (which is typically p(t) vs. x(t)).

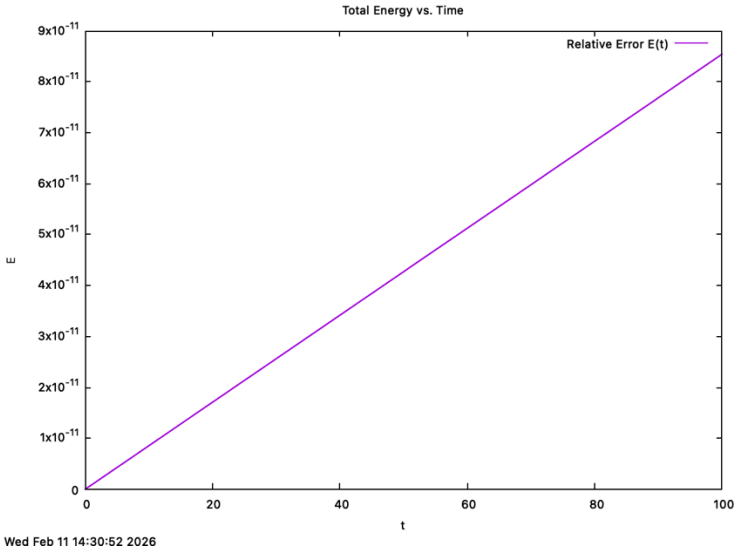
2. Wouldn't it be convenient to generate all four plots at once in separate files? Load "diffeq_oscillations_all.plt"! (Check the warning in the file on what to do if you get an "unknown or ambiguous terminal type" error message!)

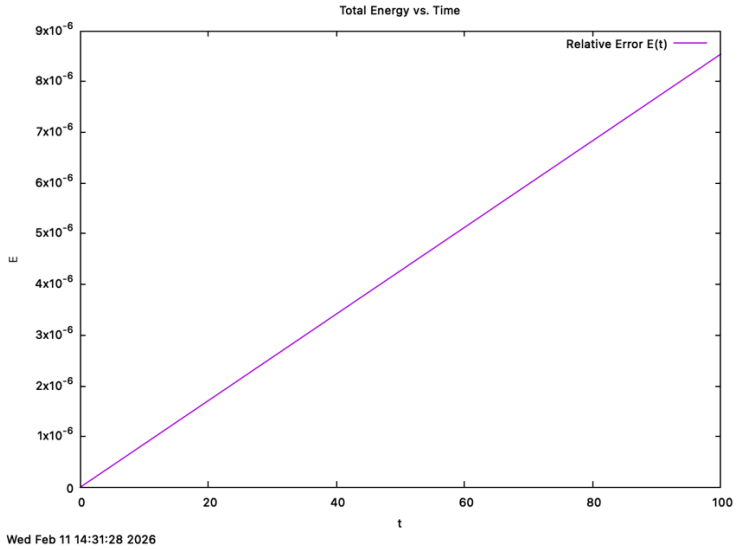
   →Done.

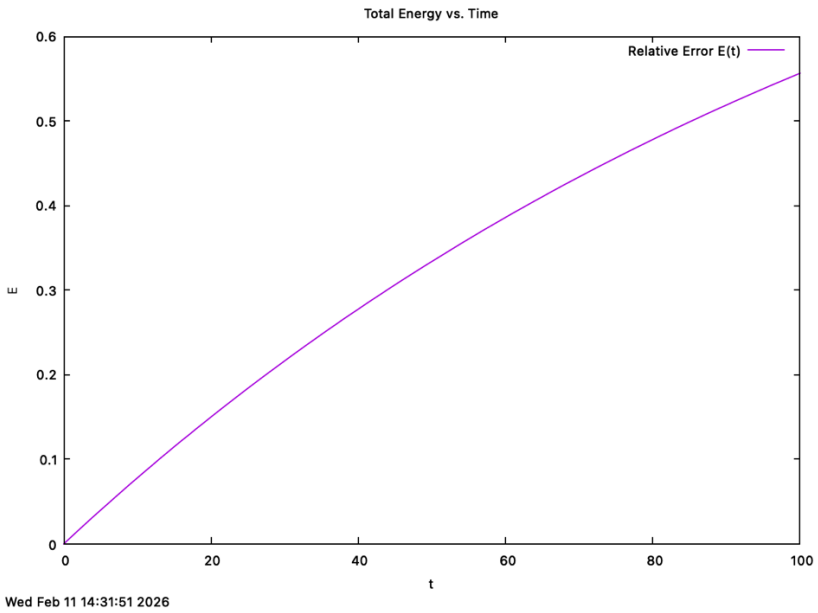
3. It's always a question whether or not you have coded a problem correctly, so you should always seek ways to check your results. One possibility is if we have a known solution. This works for p=2 (simple harmonic oscillator). What about other p? Another check is to identify a quantity that shouldn't 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?

→ We want to plot total energy (specifically, relative error of the total energy) as a function of time, as that should be invariant. We make the plot (below) and confirm that. The next two plots are for h of 10x larger and 100x larger. We can see how increasing the step size worsens the inherent conservation of energy, which is because the program doesn't know that it needs to conserve that, and the larger steps cause more drastic changes. We can find a reasonable h to use by iterating over multiple values of h and seeing the largest step size where the average total energy has the least relative error.



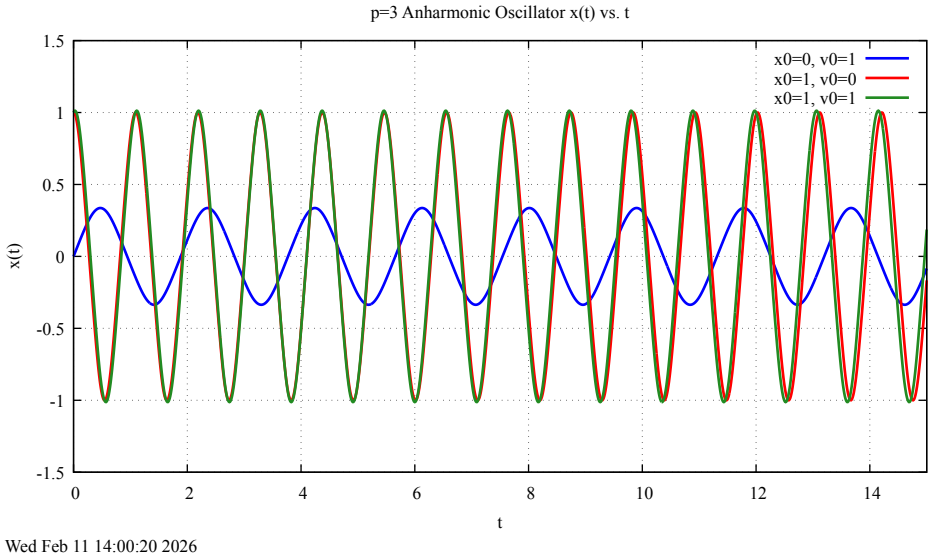


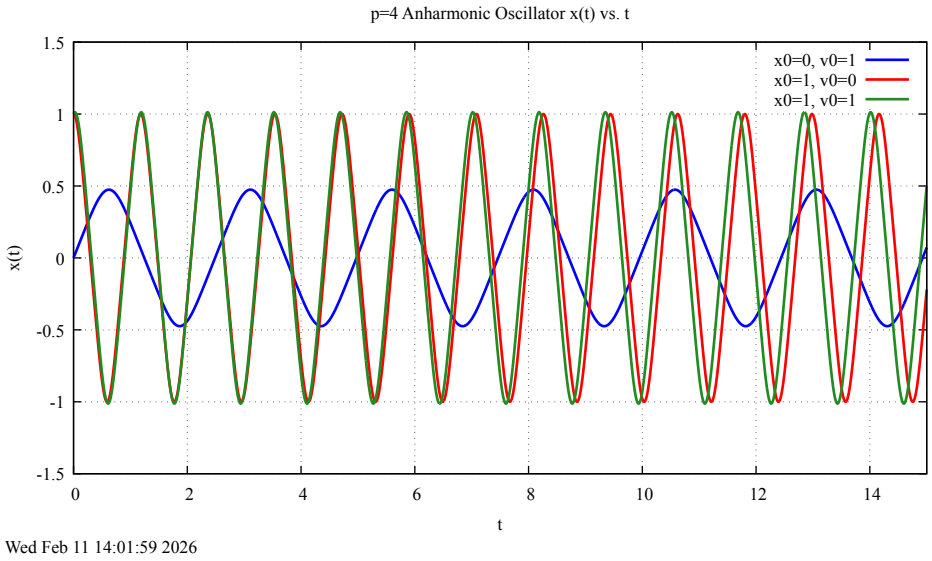


(The "plot_skip" parameter indicates how often a point is written to the output file. So plot_skip=10 means that every 10 points is output.)

4. Verify that different amplitudes (e.g., different initial conditions determined by x0 and v0) lead to different periods for an anharmonic oscillator (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? Make sure you try different values of p, since they might lead to different rules!*

→Below are two plots for (x0,y0)=(1,0), (0,1), and (1,1), with p=3 and p=4, respectively. The qualitative rules appears to be that a larger amplitude means a shorter period. This likely comes from the restoring force F being defined by a potential via -dV/dx for V(x) proportional to |x|^p, so the magnitude of the force ends up being roughly |F|= C*|x|^(p-1) for some constant C. This, if you double the amplitude x, the force can increase by a cubic or quartic factor. Then, the force and thus the acceleration grows much faster than the distance the particle has to travel, so it completes a cycle in less time.





5. 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?*

→For the p=2 case, our natural frequency w0=sqrt(k/m) is roughly 2*pi or 6.28. For w_ext=1, we are much less than resonance (w_ext roughly equals w0), so we see a beating behavior, where the oscillator itself oscillates, but also oscillates out of phase with the driving force. At w_ext=6.28, we're close to resonance, so we see the oscillator oscillate in-phase, so we get this motion where the oscillator ramps up to the driving amplitude, creating this spiral-looking phase space plot. At w_ext=3.14, we're at half of the natural frequency, finding a steady-state solution but it's non-resonant, causing some points where the driving force is effectively being

"cancelled out". For p=3, we do not see any resonance for our checked values, even though we may find steady-state solutions. This is because the w0=sqrt(k/m) holds only for the harmonic oscillator of p=2, and the anharmonic oscillator has a different w0, which can be seen by the spiraling and complex phase space plots for p=3.

C) Adding Damping

Real-world systems have friction, which means the motion will be damped. The Session_07 notes have a list of three simple models for friction. We'll implement viscous damping: $F_f = -b*v$, 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?*

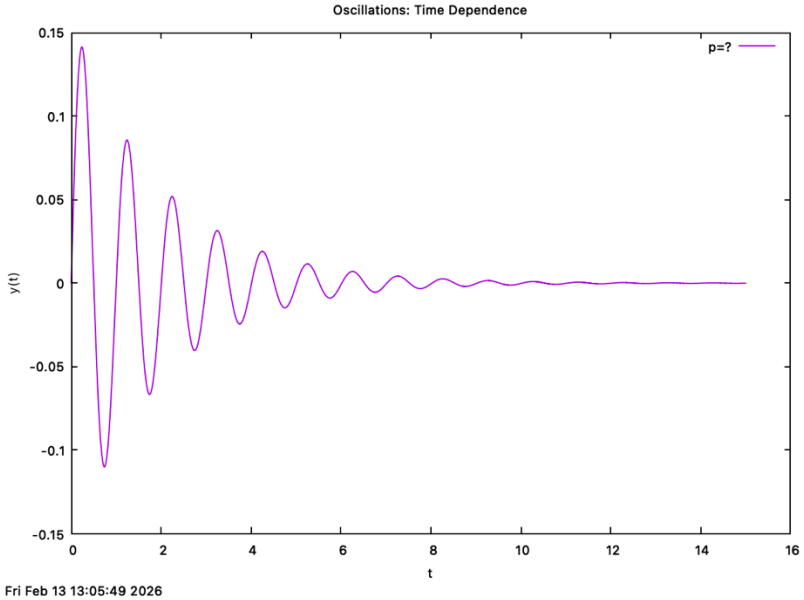
→It does work. Sinec "rhs" hasn't been modified yet, this parameter doesn't do anything yet.

2. Modify the "rhs" routine to include damping (you're on your own here!). *What did you add?*

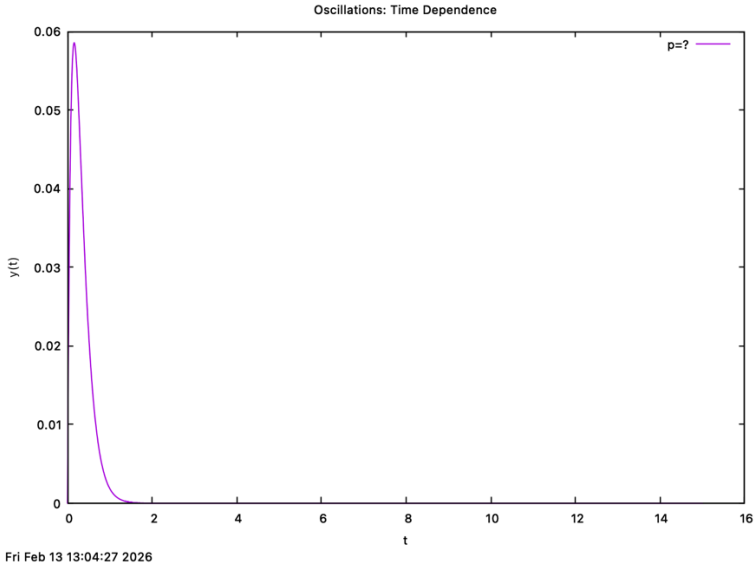
    →I added a F_restoring variable to clean up the logic, and then added the term b*y[1] which represents the damping term on the RHS when returning the result for i==1.

3. 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.
4. *Identify the three regimes described in the Session_07 notes: underdamped, critically damped, and overdamped.*

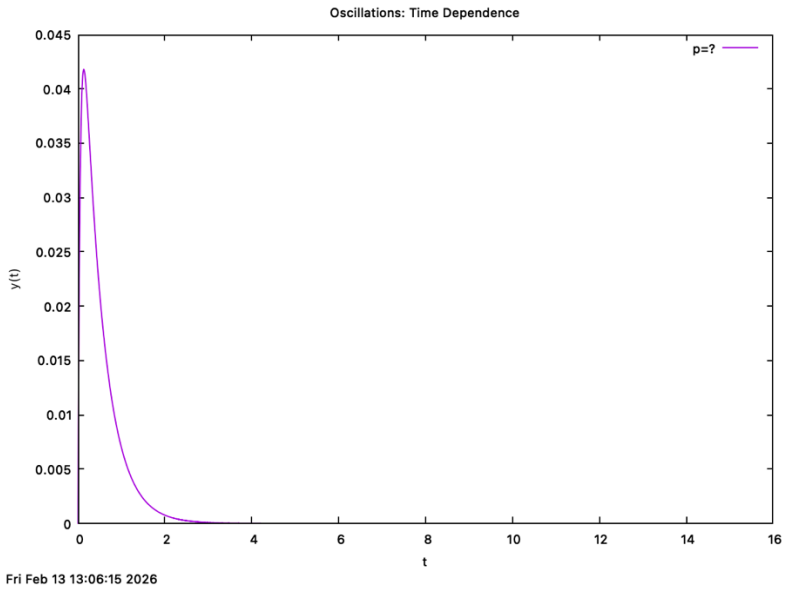
    →For p=2 from analytical analysis, we expect critical damping to occur at b=2sqrt(m*k), which for our case is about 4pi. From this we make the following plots: underdamping (b<4pi, we chose b=1), critical damping (b=4pi), and overdamping(b>4pi, we chose 20). We see that the critically damped regime has the quickest decay.



Fri Feb 13 13:05:49 2026



Fri Feb 13 13:04:27 2026

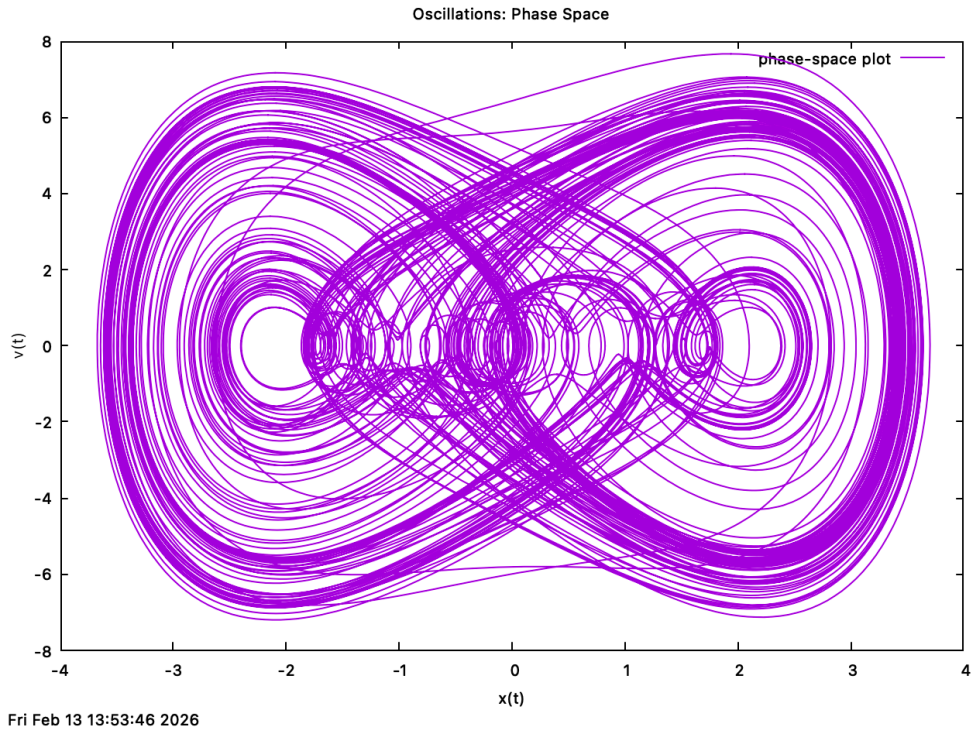


D) EXTRA: Looking for Chaos (Part I)

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_07 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 a combination of parameters for the non-linear oscillator that produce a pictures like the chaotic one in the notes?* Provide the parameters you used and include a phase space plot. Make sure to set t_end to something large (e.g., 400), so that you are not only looking at transients. Also make sure to think about the necessary conditions for chaos when selecting your parameters and to turn on some damping (without that you will see transients forever).

→ We used the parameters m=1, k=1, p=4, f_ext=10, w_ext=1, x0=0.1, v0=0, t_max=500, and b=0.1. This gets us the following chaotic multiple-attractor result:



E) Reflection

Write down which of the six learning goals on the syllabus this worksheet addressed for you and how:

→This session really helped me understand physics better by putting physics concepts like harmonic oscillators, damping, and chaos directly into computer code (especially parts C and D). I also have a better understanding of physics through playing with numerical solutions in parts C and D, and with the animations we searched for the underdamping/overdamping before starting the session.