PHY480: Activities 11

Online handouts: Pendulum power spectrum graphs, listings of ode_test_class.cpp and classes, multifit_test.cpp and multimin_test.cpp.

In this Activities, we'll take a look at nonlinear least-squares fitting with GSL.

Your goals for today:

- Take a quick look at some pendulum power spectra.
- Try out and extend the adaptive ode code with classes.
- Try out nonlinear minimization on a demonstration problem.
- Try out nonlinear least-squares fitting on a demonstration problem.

Please work in pairs (more or less). The instructors will bounce around and answer questions.

Pendulum Power Spectra [Activities 10 Follow-up]

Look at the handout with the three rows of plots showing the time dependence of a pendulum on the left and the corresponding power spectrum on the right plotted in terms of frequency = 1/period.

1. *How can you most accurately determine the single frequency in the first row from the graph on the left?*

We can determine the frequency by dividing the total number of times the wave repeats (number of peaks) by the total amount of time sampled.

2. Identify (and write down) two of the frequencies in the second row *from the plot on the left*. Do they agree with the plot on the right?

I got 22/200 and 8/200, or frequencies of 0.04 and 0.11. These agree with the plot on the right, which I assume is generated by a Fourier Transform.

3. *What is the characteristic of the power spectrum in the third row that is consistent with a chaotic signal?*

There are many similar frequencies with equal weights in the power spectrum. I imagine that small changes in the initial conditions would cause major changes in the trajectory of the pendulum in this case.

Multidimensional Minimization with GSL Routines

The basic problem is to find a minimum of the scalar function f(xvec) [scalar means that the result of evaluating the function is just a number] where xvec = (x_0, x_1, ... , x_N) is an N-dimensional vector. This is a much harder problem for N > 1 than for N=1. The GSL library offers several algorithms; the best one to use depends on the problem to be solved. Therefore, it is useful to be able to switch between different choices to compare results. GSL makes this easy. Here we'll try a sample problem. For more details, see the online GSL manual under "Multidimensional Minimization" (linked on 6810 page).

The routines used here find *local* minima only, and only one at a time. They have no way of determining whether a minima is a global one or not (we'll return to this point later).

1. Look at the test code "multimin_test.cpp" and compare it to the online GSL documentation under "Multidimensional Minimization" (there is also a handout copy). Identify the steps in the minimization process. *What classes might you introduce for this code?*

A system class that would take in any gsl multimin function and iterate our algorithm until it hits a certain number of iterations or gets a minimum within tolerance. This would really clean up the second half of the code. One might also add a function class to make defining the gsl multimin function more clean.

2. Compile and link the program with "make_multimin_test" and run it. Your results may differ from what is in the comments of the code. *Adjust the program so that your answer is good to 10^{-6} accuracy. What is it that is found to that accuracy? (E.g., is it the positions of the minimum or something else?)*

My results with a tolerance of 10^-4 agreed with the comments in the code ((x,y) = (1,2), value of f((1,2)) = 30). After changing the tolerance to 10^-6, the program found the minimum at the same place.

3. Modify the function minimized so it is a function of three variables (e.g., x, y, z), namely a three-dimensional paraboloid with your choice of minimum. Change the code so that the dimension of xvec is a parameter. (Be sure to change ALL of the relevant functions.) *Is the minimum still found?*

I changed the code to find the minimum of the parabola $10(x-1)^2 + 20(y-2)^2 + 5(z-3)^2$ since I am not very original, the minima being at (1,2,3). I started the minimizer off at (5,7,9). The minimum was still found, with the answer converging after 17 steps.

4. *What algorithm is used initially to do the minimization? Modify the code to try steepest_descent and then one of the other algorithms. Which is "best" for this problem?*

The current algorithm being used is the Fletcher-Reeves conjugate gradient algorithm. I couldn't tell you what that is, but it's what the code is using.
Using the Steepest Descent Algorithm, the minimizer got stuck very close to the minimum, but didn't make it all the way (ended up at the point (0.98, 2.06, 3.23).
I tried out the gsl_multimin_fdfminimizer_vector_bfgs2 algorithm on a whim and it found the exact minimum in a scary three steps. This algorithm seems incredibly well suited to this kind of function, although I don't know what it is (name is the Broyden-Fletcher-Goldfarb-Shanno)
The last algorithm listed for functions with derivatives was the Polak-Ribiere conjugate gradient algorithm (gsl_multimin_fdfminimizer_conjugate_pr). It found the exact minimum in 17 steps.

From this very naïve analysis, I'd say the Fletcher-Reeves and the Polak-Ribiere perform well for these kind of smooth paraboloid problems, but are much slower than the Broyden-Fletcher-Goldfarb-Shanno algorithm which also finds the exact minimum. The Sttepest Descent Algorithm is the least accurate of them all.

Nonlinear Least-Squares Fitting with GSL Routines

Using the nonlinear least-squares fitting routines from GSL is similar to using the GSL minimizer (and involves a special case of minimization). Your main job is to figure out from the documentation (see 6810 page) and this example from GSL (only slightly modified from what you will find in the documentation) how to use the routines. Here we briefly explore the sample problem, which also illustrates how to create a "noisy" (i.e., realistic) test case ("pseudo-data").

The model is that of a weighted exponential with a constant background:
$y = A\, e^{-lambda\ t} + b$
You'll generate pseudo-data at a series of time steps t_i (in practice, t_i is taken to be 0, 1, 2, ...). Noise is added to each y_i in the form of a random number distributed like a gaussian with a given standard deviation $sigma_i$. We'll revisit the GSL functions that generate this random distribution in the next Activities; for now just accept that it works.

1. Take a look at the multifit_test.cpp code (there is a printout), run it (compile and link with

make_multifit_test) and figure out the flow of the program. *What is the fitting ("objective") function to be minimized? What role does sigma$_i$ play?*

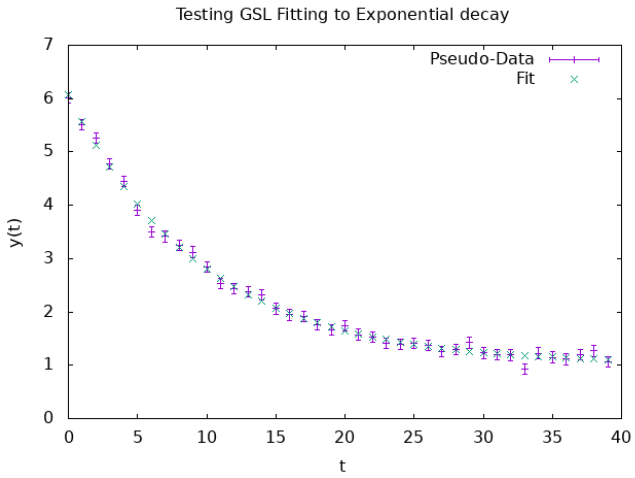
sigma$_i$ is the noise in our "data." Increasing sigma$_i$ will increase the amount.
The objective function is given above as Y = A* e^(-lambda * t) + b. We are trying to minimize the the sum of squared residuals between each point on the fit and the given data.

2. *What is the "Jacobian" for? (Check the online documentation for the answer!)*

As best I can make out from the online documentation, the Jacobian is used to tell how sensitive the error residuals are with respect to each parameter (A,lambda, and b for our case). I assume the fitting algorithms use this information to converge on a value more quickly.

3. *Modify the code so that you can plot both the initial data and the fit curve using gnuplot. Look up how to add error bars for the data in gnuplot. [Hint: Try "help set style", "help errors", and "help plot errorbars" for some clues.] Did it work??.*

I added error bars equal to the specified sigma:



4. The covariance matrix of the best-fit parameters can be used to extract information about how good the fit is (see notes). *How is it used in the program to estimate the uncertainties in the fit parameters? How do these uncertainties scale with the magnitude of the noise? Do they scale with the number of time-steps used in the fit?* (I.e., if you change the "amplitude" of the gaussian noise, how do the errors in the fit parameters change?)

Increasing the noise increases the covariance in the parameters. I tried out a sigma of 0.1, 1, 10, and 100. With a sigma of 100, no solution could be reached (which makes sense, we have noise an order of magnitude larger than the objective A value). For 0.1, 1, and 10, the entries of the covariance matrix of the best-fit parameters increased with several orders of magnitude each time I increased the order of magnitude of the error. However, I am not sure of the exact relationship.
Holding sigma at 0.1, I ran the fit with N=40,400,4000. The entries of the covariance matrix of the best-fit parameters decreased as N increased, though again I'm not sure of the exact relationship.

GSL Adaptive ODE Solver Revisited

In Activities 10, we took a quick look at ode_test.cpp, which implemented the GSL adaptive differential equation solver on the Van der Pol oscillator. Here we look at a rewrite that uses classes.

1. The details of ode_test_class.cpp and the Ode and Rhs classes are described in detail in the

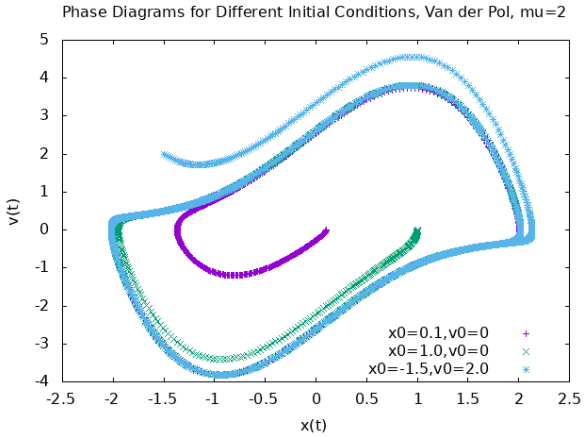
Activities 11 notes. Look at the printout while you go through the notes. *What questions do you have?*

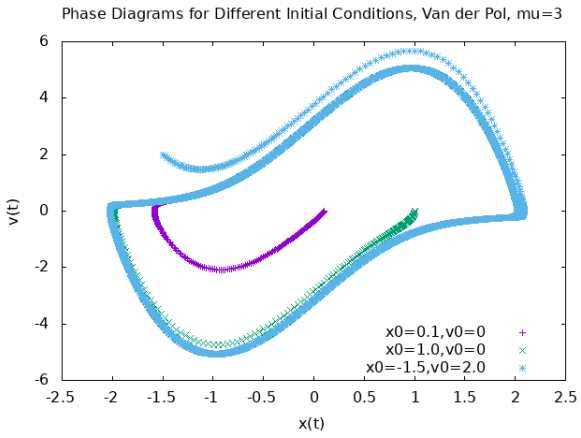
It makes sense, just a more streamlined version of what we did in activity 9/10.

2. Add two additional calls to evolve_and_print so that all three initial conditions from Activities 10, [x0=1.0, v0=0.0], [x0=0.1, v0=0.0], and [x0=-1.5, v0=2.0], are generated with the same run. *Did the three output files get generated? (You can try plotting to check correctness.)*

The output files were made and things seem to be in order.

3. Add another instance of the Rhs_VdP class called vdp_rhs_2 with mu=3 and generate results for the same initial conditions. *Plot these. Is it still an isolated attractor?*





After changing mu to 3, we don't see much difference in the trajectories. This still looks an isolated attractor.

4. [Bonus. Come back to this if you finish everything else.] Add another class Rhs_Pendulum that implements the pendulum differential equation with natural frequency omega0 and a driving force with amplitude f_ext and frequency omega_ext. (You will want to modify or replace evolve_and_print.)