PHYS 331 – Introduction to Numerical Techniques in Physics

Homework 5: Spline Interpolation, Least-Squares

Due Friday, Sept. 29, 2017, at 11:59pm.

Problem 1 – Lagrange Method (10 points)

Do Problem 9 of Problem Set 3.1 (variation in density of air with elevation). This is pen-and-paper only (no Python). Show your steps (try to do some simplification without a calculator first).

Problem 2 – Interpolation in Two Dimensions (20 points)

Recall that the reading material for this topic is in Numerical Recipes which is uploaded to the Sakai lecture notes. Consider the 2-dimensional function:

$$f(x,y) = x^2 - y^2 = z \tag{1}$$

(a) First, let's sample this function over a relatively coarse mesh of discrete values in x and y, then display it on an intensity plot using the `plt.imshow` function. Consider a range of (-1,1) in both x and y with a coarse step size of 0.2. In order to do this, you may consider looping over x and y values individually. Hint: I often find it helpful to "pre-allocate" an array by using the `np.zeros` command, *i.e.*, creating an array of zeros that I can fill in later. Get help with this step if needed, because you will need the strategy developed here for writing your bilinear interpolation function in the steps to follow.

Warning: The "x" and "y" axes in terms of how they are indexed in a 2D array are flipped, *i.e.*, `my_ary[yi,xi]` would index row `yi` and column `xi`. Also, note that the y-axis itself is, in general, flipped, *i.e.* low values of the row number appear at the top of the image, and high values appear at the bottom. For now, we will ignore these effects, unless you wish to do the extra credit task. You do not have to worry about axis labels or getting the tick values to display the real values of x and y in this problem.

(b) Now, write a function that can perform the interpolation over a finer mesh using 2D bilinear interpolation. Consider a mesh size of 0.01 to sample the interpolated function. Write a function of the form `zsamp = bilinear2D(xsamp,ysamp,xdata,ydata,zdata)`, where:

- `xsamp` and `ysamp` are vectors (numpy 1D arrays) containing the x and y values at which to sample your interpolation function
- `xdata` and `ydata` are vectors (numpy 1D arrays) containing the original x and y values of your data
- `zdata` is the 2D array containing the data – this would be the 2D array that you plotted in part (a).
- `zsamp` is the output 2D array containing the interpolated function sampled at the `xsamp`, `ysamp` points.

This will not be simple, but I believe you can do it! Try to plan out your function first, thinking about the steps needed to do the interpolation. Remember, the first step is to find the x_n, x_{n+1} and y_n, y_{n+1} points of the data that bracket the point you want to sample. So, thinking about using loops to step through your `xsamp` and `ysamp` points, at each point you need to search first for these bracketing data values.

Warning: One thing that will definitely cause problems is if you try to sample the function outside of the original data space. So, make sure that your `xdata`, `ydata` really span all the way from -1 to 1, and that the `xsamp`, `ysamp` values that you pass to your function do not extend beyond that interval.

(c) Finally, sample your original function on the finer mesh of 0.01, over the same interval, and display the resulting data using `imshow`. What do you notice about the result compared to the original data? Make sure both your outputs from part (a) and part (b) display when executing the script (you may need to use `plt.figure` and `plt.show` so that the images don't over-write one another).

Extra credit (up to 5 points): The goal here is to modify your plots to display the x and y axes in the correct orientation, and label the correct values of x and y on the sides of the plot over the intervals from -1 to 1. In order to test that your modifications work correctly, let's use a new function, `f2`, defined as:

$$f_2(x) = x^2 - x + y^2 + y \tag{2}$$

that lacks the symmetry of the previous function. Display the result of this function interpolated over the fine mesh (as in part (c)), with the correct axis orientations, labels, and tick marks. Also, display a scale bar that indicates how the z values correspond to the color scaling used in the plot. Warning: the instructor hasn't done this problem yet, so doesn't know how hard it might be.

Problem 3 – Nonlinear Least-Squares Fitting: an example from Optical Spectroscopy (20 points)

In PHYS 118 you have already done least-squares fitting of polynomial functions or functions that can be linearized (such as linearizing exponentials by taking a log of both sides). However, one of the most useful general skills is the ability to fit experimental data to arbitrary, nonlinear functions. As with any computational task, there are tricks and no one "right" way. Here we will explore one method that works for most such tasks, and you will already be more experienced than many first-year graduate students. Be patient – the problem looks long, but I will walk you through this step by step.

Many atomic and molecular energy level transitions result in the emission of photons at specific peak wavelengths λ, where the lineshape (the shape of the optical spectrum) is described by a Lorentzian function:

$$L(\nu) = \frac{1}{\pi} \frac{\frac{1}{2}\Gamma}{(\nu - \nu_0)^2 + \left(\frac{1}{2}\Gamma\right)^2} \tag{2}$$

where ν, the wavenumber, is defined as the reciprocal of the wavelength, $\nu = 1/\lambda$; the peak wavenumber, $\nu_0 = 1/\lambda_0$, is where the lineshape is at the maximum; and Γ is the full-width at half maximum, *i.e.*,

$$L\left(\nu \pm \frac{\Gamma}{2}\right) = \frac{L(\nu_0)}{2} \tag{3}$$

The lineshape function is already normalized such that its integral = 1. The wavenumber is a convention used in optical spectroscopy as it is more convenient than the optical frequency ($\omega = 2\pi c/\lambda$) because it can be computed more simply from $1/\lambda$. Somewhat strangely, it is given in units of cm^{-1}.

In the laboratory, you collect spectral data of a mixture of gases and note that there are two spectral lines. As such, you wish to fit your data (the optical intensity), $S(\nu)$ sampled at discrete values of ν in units of cm^{-1}, to a function of the form:

$$S(\nu) = c_1 L_1(\nu) + c_2 L_2(\nu) \tag{4}$$

where c_1 and c_2 are constants that indicate the relative strengths of the transitions, and the peak wavenumbers ν_{01} and ν_{02} associated with Lorentzian lineshapes L_1 and L_2 will give you needed information to determine the energy levels involved in the transitions. Note that L_1 and L_2 also each have an associated value of Γ_1 and Γ_2, respectively. Below you will learn to import your data into Python, and perform a nonlinear least-squares fit in order to extract the parameters of interest.

(a) The spectral data has been provided to you as a comma-separated-values (CSV) file, *HW5p3data.csv*. The CSV file format is a text file** containing a pair of (ν, $S(\nu)$) numbers in each line, separated by a comma. I recommend opening and examining the contents of the file in a program like Excel.

Use the numpy function `loadtxt` to import this data into an array. First, check out the info page on this function: https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.loadtxt.html Note that one parameter is required to be passed to this function (*fname*), as well as several optional parameters that can be passed (this includes anything that is already set = to something else, indicating that it has a default value if not used). In particular, you will want to use the "*delimiter*" parameter to tell Python that commas are being used to separate (delimit) each set of data values. What is the size (height x width) of the resulting array?

** A "text" file means that the characters are encoded using the ASCII format (see, for example, http://www.asciitable.com/), which is a very common format used by simple, non-formatted editing programs like notepad. Your Python scripts are also stored in this format.

(b) Now that you have imported your data into an array, plot it. Recall, the plot function requires separate vectors for your "x" and "y" values to be plotted, but the data matrix that you just loaded isn't natively in this format. Remember what your TA told you about how to call certain rows or columns of an array? (All lab presentations are in Sakai just like the lecture…) Since you will be using these later on, assign your "x" values to a new array called "`v`" to represent your wavenumber values, and your "y" values to a new array called "`Sv`" to represent your optical intensity measurements at each v.

(c) Now, write a function `ModelSpectrum(c1,c2,v01,v02,g1,g2,v)` which takes the input wavenumber, `v`, and all of the other input parameters needed to define your spectrum S(v) of Eq.(3), and outputs the value of $S(\nu)$. Take an educated guess at the input parameters, and plot the function over the same range in ν as your data, in a separate plot. It should look like two peaks, although the heights, widths, and positions will not match your data. Adjust your initial guess as needed to make it look reasonably close to the input data.

(d) Now, copy-paste your function into a new function, `ModelSpectrum2(x,v)`, where `x` is now a vector containing all of the parameters `c1, c2`, etc. Note that you will now need to modify the computations within your function to call individual elements of `x`. For example, the first element of `x`, `x[0]`, would replace `c1` in your computation, `x[1]` replaces `c2`, and so on. Plot the output of the function over the same range of `v`, using the same values of the input parameters as in part (c). To do this, first, define a vector `x0` as a numpy array containing your initial guess values `(c1, c2, v01, v02, g1, g2)`, then plot `ModelSpectrum2(x0,v)`. It should look the same as that in part (c) before proceeding.

(e) There is still one more step before you can to the fit! Copy paste ModelSpectrum2 into yet a new function, `ModelSpectrum3(x,v,Sv)`. The idea here is to now have it return values that should be zero. In other words, after computing the estimated value of S(ν) based upon `x` and `v`, you subtract the data value `Sv` from the function. (Example: if $f(x) = y = x^2$, you would define $f_2(x,y) = x^2 - y$). This is because the least-squares algorithm is a minimization algorithm. So, you really only have to do a minor modification here. Try plotting `ModelSpectrum3(x0,v,Sv)`. Is it close to zero? Maybe not yet, because we haven't optimized the parameters in the vector x.

(f) Finally, look up the scipy.org help page on the function `scipy.optimize.leastsq()`. It's not terribly helpful, although there is an implementation example at the bottom. But, with the guidance above, you are now pretty much ready to use this! First, `import scipy.optimize`. You have already defined the function to be minimized, `ModelSpectrum3`. You have already established a reasonable starting point for your parameters, `x0`. And you already have your data, `args = (v, Sv)`. When you run the function, what does it return?

(g) Let's extract the fitted parameters and make sure they work. Store the output of the `leastsq` function in part (f) into the variable `res`. Note that it is a tuple of size 2. Assign the first element of the tuple to a new variable, `x1`. What is the data type and size of `x1`? Now plot `ModelSpectrum3(x1,v,Sv)`. Is it close to zero?

(h) Finally, overlay your original data (plotted without any interpolating lines) with the best-fit curve using `ModelSpectrum2(x1,vmesh)`. Note that it makes more sense to use a `vmesh` with much smaller spacing than the data `v`, because your best-fit curve is now a well-defined function at all v. Do they match reasonably well? Does the best-fit curve go exactly through the data points, or only approximately? What are the best-fit peak wavenumbers, `v01` and `v02`?