Nikhil Khosla

PHYS265

Teuben & Hall

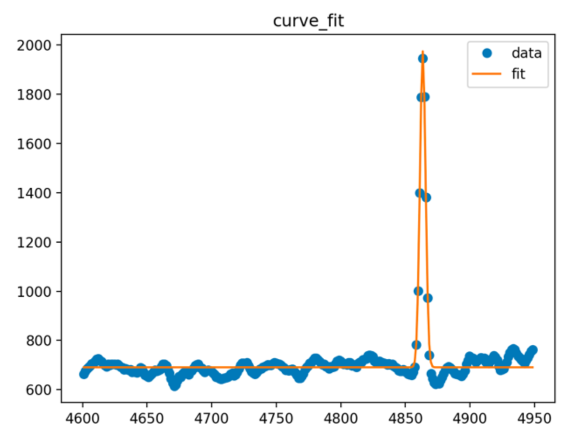
May 2, 2024

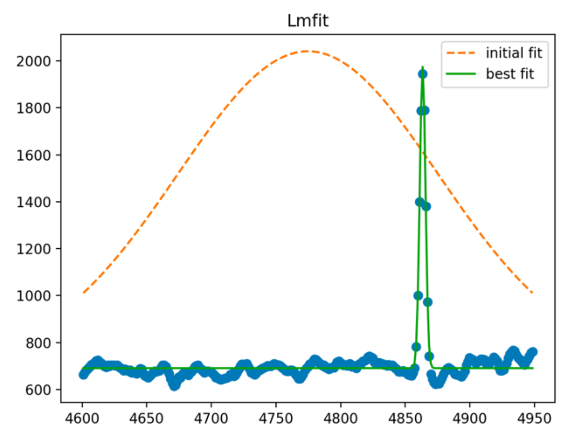
PHYS 265 Code Project

For this coding project, I chose the Lmfit package. This package is described as a curve-fitting and non-linear least-square minimization package for Python. It serves to give fitting results and plots of the fits of datasets in a non-linear fashion. Lmfit was created 8 years ago initially inspired by and named after the Levenberg-Marquardt method from scipy.optimize.leastsq. However, Lmfit improves on multiple aspects of it such as: improved estimation and curve fitting, and different parameter options. It is still maintained by the original author as recent as two weeks ago (fun fact: this is also one of the authors of matplotlib in 2005). The installation of Lmfit is quite simple, with only a few steps. Overall, it shouldn’t take more than a couple minutes to install. Learning how to use it is a bit more complex, however, does not take much time to understand as well. It helps to look at the example gallery that shows the various uses of Lmfit. For example, Lmfit can be used for fitting two-dimensional peaks, fitting multiple datasets, using boundaries for fitting, or calculating confidence intervals. The source code is available to everyone, as it is publicly posted on GitHub. In my examples provided below, I simply used python to write and execute the code. Lmfit is a pure python package and does not need accompanying C/C++/etc. To install, all that is needed is a simple pip install and git clone. The input to Lmfit is a dataset which you can input directly or create your own randomized one if you don’t already have a dataset. In my examples below, I have shown one example where the dataset was already provided (in the Hbeta function code) and another example where the dataset is randomly made by the user. The output then is the parameters of the function which defines the fit of the function. The code provides no unit tests, regression, or benchmarking, however, by manual inspection of the dataset and verifying parameters based on such, you can feel confident in the results of the code. This code mainly relies on numpy and scipy in order to carry out simple arithmetic and data calculations. The package produces figures using matplotlib where the plots of the dataset and fit curve are plotted. This package also does not give a preferred citation method.

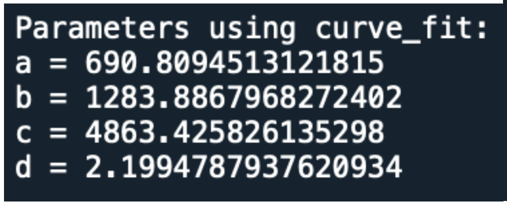
Code example 1:

In code example 1, the expression model being used is that of the H Beta line which refers to a hydrogen emission line which corresponds to the transition of an election from n=3 -> 2 in a hydrogen atom. Here, a dataset was provided by Dr. Teuben which was loaded into a txt file. The formula used in the code is that of a gaussian function a+b*exp(-(x-c)**2/(2*d**2)) where a is the baseline of the function, calculated by taking the mean of the y-values, b is the amplitude of the Gaussian peak found by subtracting the minimum from the maximum y-value of the dataset, c is the center of the x-values found by taking the mean of x-values, and d is the standard deviation which can be found by using np.std of the x-values. I used both curve_fit and Lmfit with this dataset to show the similarities and differences that they achieve. As shown below, the graphs look nearly identical, with the Lmfit graph just also having an initial fit line.

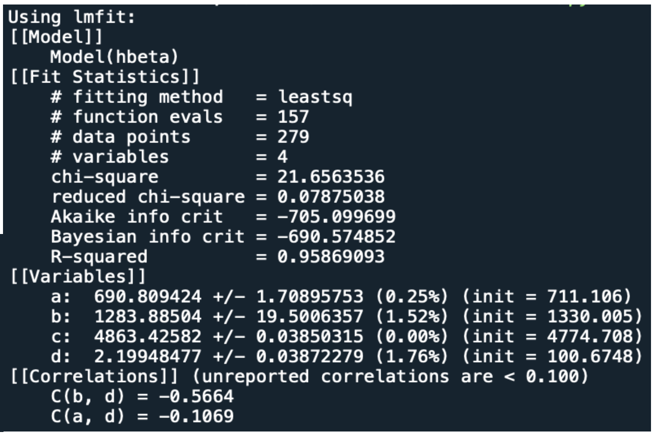




However, this does not mean that the packages are identical. The strength of Lmfit is the data provided in the terminal after running the code. When using curve_fit, simply the best fit parameters are printed out to the terminal. However, when using Lmfit (as shown below), the printing of a simple function res.fit_report() outputs to the terminal both the parameters and tons of other information about the data such as: the chi-squared value, # of data points, R squared value, and more. This highlights the easy yet effective use of Lmfit as opposed to a simple curve_fit package.



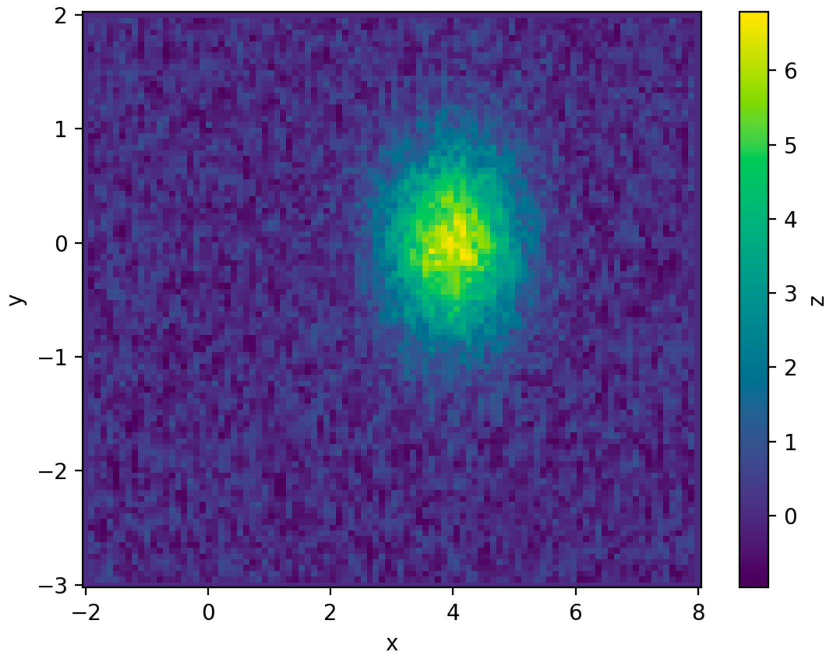
```
Parameters using curve_fit:
a = 690.8094513121815
b = 1283.8867968272402
c = 4863.425826135298
d = 2.1994787937620934
```



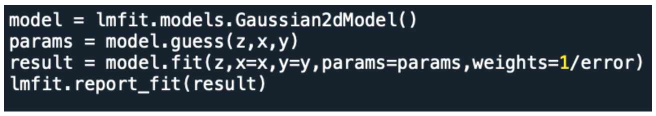
```
Using lmfit:
[[Model]]
    Model(hbeta)
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 157
    # data points      = 279
    # variables        = 4
    chi-square         = 21.6563536
    reduced chi-square = 0.07875038
    Akaike info crit   = -705.099699
    Bayesian info crit = -690.574852
    R-squared          = 0.95869093
[[Variables]]
    a:  690.809424 +/- 1.70895753 (0.25%) (init = 711.106)
    b:  1283.88504 +/- 19.5006357 (1.52%) (init = 1330.005)
    c:  4863.42582 +/- 0.03850315 (0.00%) (init = 4774.708)
    d:  2.19948477 +/- 0.03872279 (1.76%) (init = 100.6748)
[[Correlations]] (unreported correlations are < 0.100)
    C(b, d) = -0.5664
    C(a, d) = -0.1069
```

Code Example 2

In code example 2, I used Lmfit in order to fit two-dimensional peaks. Here instead of being given a dataset to use, I created randomized data to be put into a grid. This helps to simulate noise and randomness of experimentation. Interpolating this data onto a grid with random points, but a set amplitude, center, and uncertainty, gives the dataset in a 2D contour fashion as shown below.

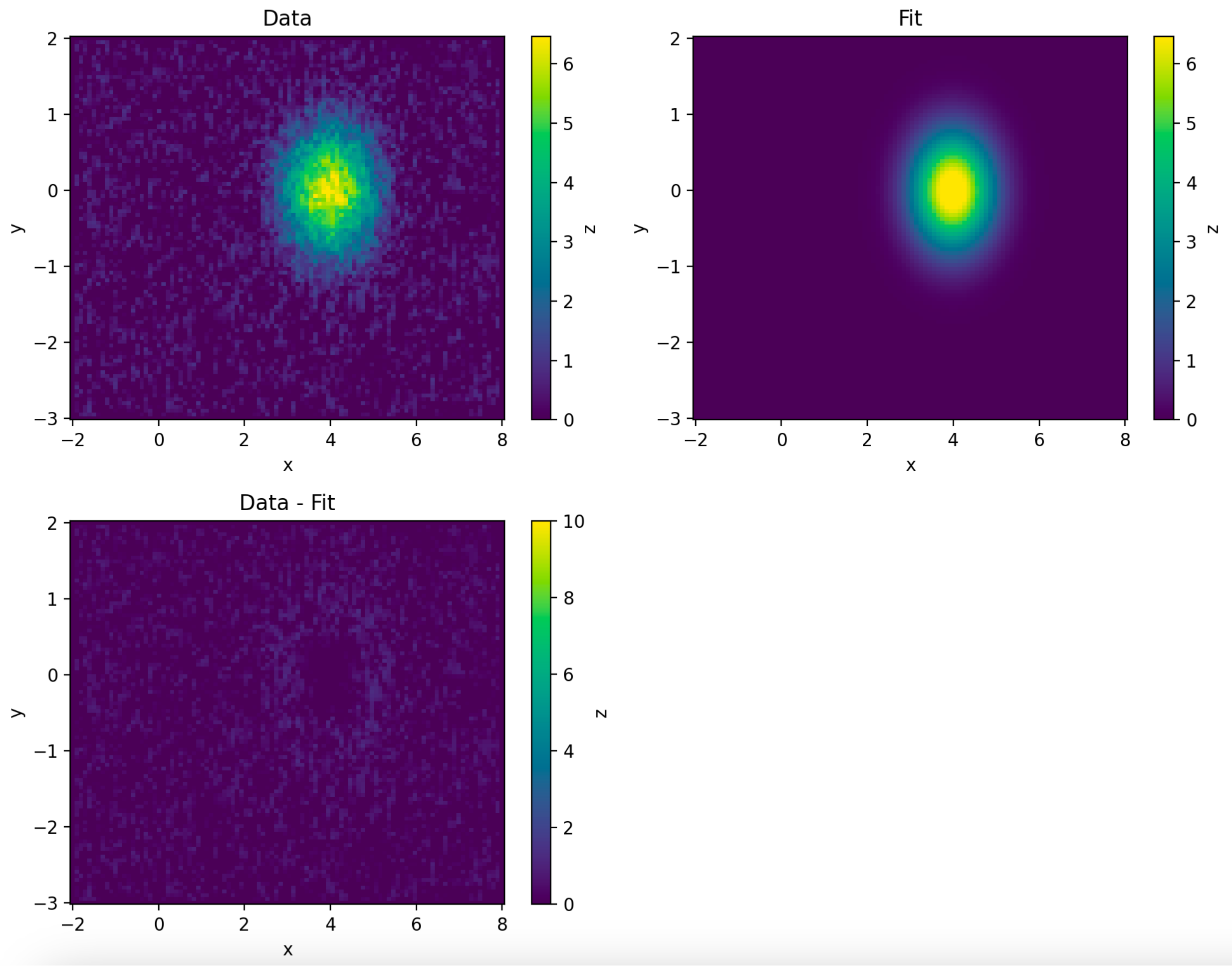


Next, I created 3 subplots where the first would plot the initial data as shown above, then the next plot would be of the fit data. This will give a more refined look to the data, giving it a two-dimensional non-linearity best fit.



```
model = lmfit.models.Gaussian2dModel()
params = model.guess(z,x,y)
result = model.fit(z,x=x,y=y,params=params,weights=1/error)
lmfit.report_fit(result)
```

Lastly, to verify that this is indeed an effective fit of the data, I subtracted the initial values from the best fit values which should theoretically cancel and remove the peak. These plots are shown below.


Data
Fit
Data - Fit
x
y
z