Modeling 8

Ryan Swope

April 20, 2021

1 Introduction to Linear Least Squares Fitting

Least squares fitting is an application of Bayesian statistics for linear systems. The idea is that the probability of an event D happening given the parameter condition c can be predicted given a set of observed data, and a model of the data that is dependent on the parameter c and the independent variable from the observed data, x. Skipping much of the theory we covered in class, the technique essentially distills to the following: Let y and x be observed data of length N. If there exists a linear relationship between x and y then a model of the system can be expressed as:

$$\mathbf{Ac} = \mathbf{y} \tag{1}$$

Where the matrices are:

$$\mathbf{A} = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{pmatrix}, \ \mathbf{c} = \begin{pmatrix} a \\ b \end{pmatrix}, \ \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

Clearly, in this model each point of the model would look like a typical linear equation, $y_i = ax_i + b$. However, sometimes this basis is insufficient for the application. Fortunately, this model can be extended to a system of M basis functions. The matrices in this case then become:

$$\mathbf{A} = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \dots & f_M(x_1) \\ f_1(x_2) & f_2(x_2) & \dots & f_M(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(x_N) & f_2(x_N) & \dots & f_M(x_N) \end{pmatrix}, \ \mathbf{c} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_M \end{pmatrix}, \ \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

The system $\mathbf{Ac} = \mathbf{y}$ can then be solved by minimizing the square of the residuals, χ^2 which is formulated as:

$$\chi^2 = ||\mathbf{Ac} - \mathbf{y}|| \tag{2}$$

For the purposes of this project, I used the `numpy.linalg.lstsq` package to solve for $\mathbf{c}$. This function takes in $\mathbf{A}$ and $\mathbf{y}$ as arguments and returns, among other things, the optimal solution for $\mathbf{c}$. To obtain the modeled data values from this, simply take the dot product of $\mathbf{A}$ and $\mathbf{c}$.

2 Iodine Injection

We are given data for Iodine concentration in the blood as a function of time, presumably after an injection. Unfortunately, this data is not readily linear. In fact, we are told to attempt to fit the function:

$$f(t) \propto \exp(-\lambda\sqrt{t}$$

This would be trivial to fit with a non-linear routine, but alas this is forbidden. Instead, we must first transform this into a linear model which we can solve and then apply the inverse transformation to get the model for the provided data. We are ignoring additive constants (natural iodine levels) for now. So, we can write the amount of iodine as:

$$I = Ae^{-B\sqrt{t}} \tag{3}$$

Taking the natural logarithm of both sides and applying some logarithm rules, we obtain:

$$\ln I = \ln A - B\sqrt{t} \tag{4}$$

. Thus, the natural logarithm of the iodine levels are linear to the square root of time!. Thus, we can input the square root of time into **A** and the natural logarithm of our iodine values into **y**. Of course, we cannot simply plot this along side our data, because the solver is going to give us values for $\ln A$ and B, because they constitute the matrix **c**. However, we can apply the inverse of the transformation we made to linearize the data by taking the exponential of the equation, or more simply we can recover the value of the constant A and plug that, along with B, directly into the model we started with.

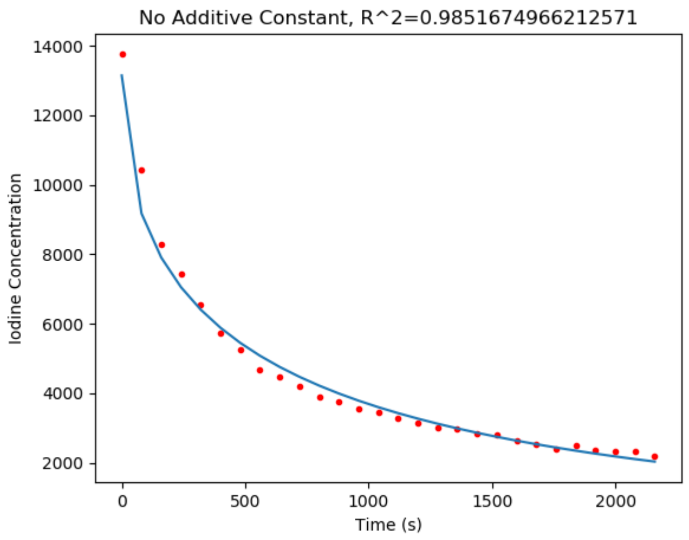

Figure 1: LLS fit of iodine levels, no additive constant

However, we know that the human body contains at least trace amounts of almost every element, so it's unreasonable to assume that there is no background or biological levels of iodine in blood. Taking this into account gives us the model:

$$I = Ae^{-B\sqrt{t}} + C \tag{5}$$

Which cannot be linearized. If you try to apply the same technique as before, you find it is impossible to separate C. So, instead we can subtract C from both sides and then take the natural logarithm of both sides to obtain:

$$\ln I - C = \ln A - B\sqrt{T} \tag{6}$$

Thus, we can solve for A and B as before, with the linear least squares solver. The difference is our **y** array now has two variables: I(t) and the constant C. To incorporate C into our model, we will have to employ a brute force method to find the best value for C, within a certain tolerance. I accomplished this with an algorithm that operated as follows: An initial C guess was made; I started with euler's number, e. Using that value, create a model using the linear least squares algorithm and calculate the residuals of that model. Then, increase and decrease C by 1, make new models with those values and compare the residuals. If C+1 was better, then restart from there, and if C-1 was better, restart from there. However, if C was still the best of the three, the step size was decreased by one half, and the operation was performed again. This was repeated until the step size well below the tolerance level. Then, knowing the value for A, B, and C of that model, the exponential model could be recovered. I determined the background level of iodine to be 758.71. Incorporating this into the model gave a fit with a better $\mathtt{R}^2$.

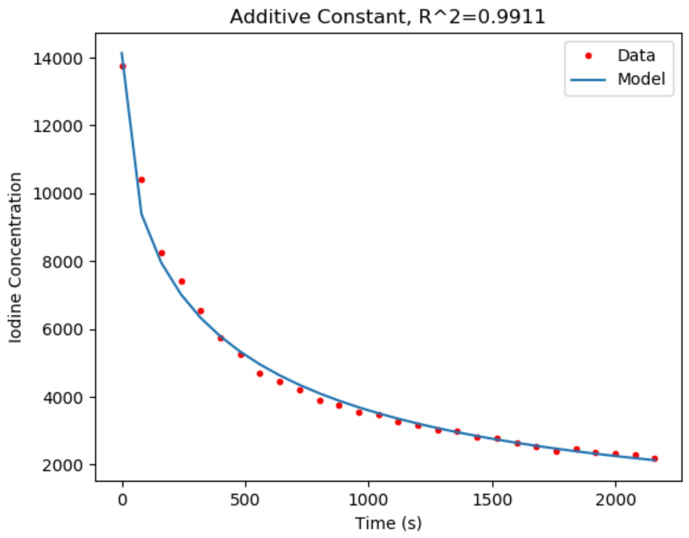

Figure 2: LLS model of iodine, with additive constant

Despite the better fit, both based on the $\mathtt{R}^2$ and by visual inspection, statistic had another idea. The mean residual value for the first model was 32.74 and the average value for the second model was 17.41. Performing a t-test gave a test statistic of 0.1937. However, for 95% confidence on a data set of 28 values, t is 2.052. Thus, we failed to reject the null that there was a difference between the models. In fact our p value was, as Obi-Wan Kenobi would say, ”off the chart” at greater than 0.40, indicating we have strong evidence to not reject the null.

3 Sigma Clipping

Sigma clipping is an important data cleansing routine in astronomy, because it eliminates extraneous points and allows astronomers to accurately measure flux values of observed objects across ranges of the electromagnetic spectrum. To sigma clip data, we fit a model to it, and calculate the variance of that model σ. We then check if data points lie within a range around our model, say between $+\eta_1\sigma$ and $-\eta_2\sigma$. If they do not, they are eliminated from the data set, the model is refit, and the data is clipped again until the data has been sufficiently cleansed. I sigma clipped the spectrum of two stars, Vega and h Tau. Instead of using the typical basis for linear functions, however, we will instead use the Legendre polynomials. These polynomials are highly orthogonal and good for fitting non-oscillatory data; as such they are used in the solution for Laplace's equations and are a good fit for the sigma clipping routine because of their higher order terms. Fortunately, the change in basis functions is trivial as discussed above.

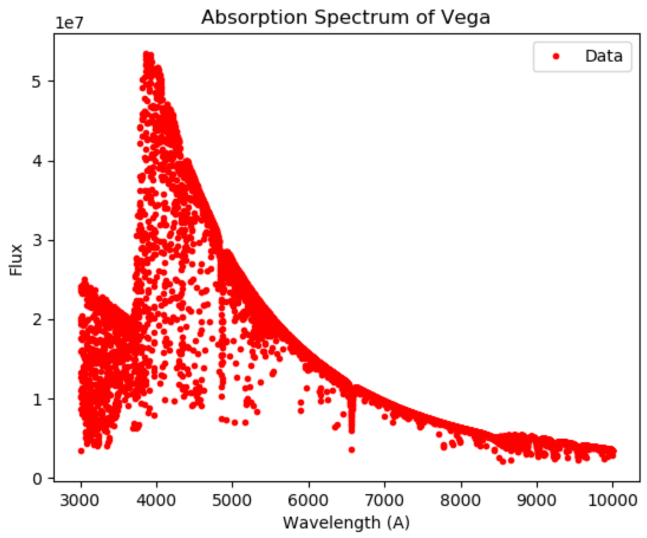

Figure 3: Unfiltered Vega spectrum

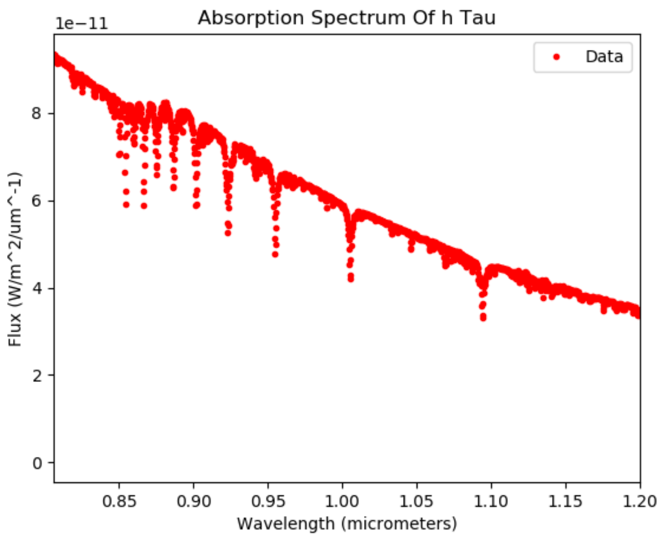

Figure 4: Unfiltered h Tau spectrum

Vega especially has a significant amount of noise in its data, which made even an initial fit difficult to make; To work around this issue I filtered the data once through with a Savitzky-Golay filter set to a window size of 99 and using 7th order polynomials. I then sigma clipped the data based on variance of the model. The lower limit of the clipping was significantly less stringent than the upper limit, so as to not cut out the very absorption lines we are trying to pick out. Once the data had been sigma clipped, the absorption lines became very clear.

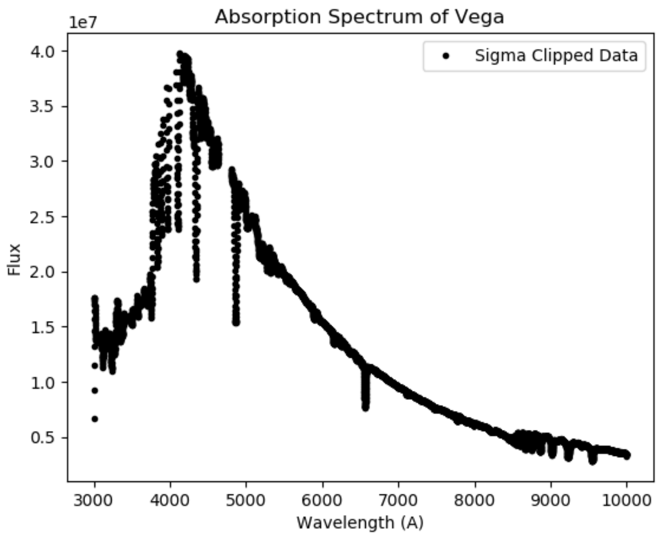

Figure 5: Sigma clipped Vega spectrum

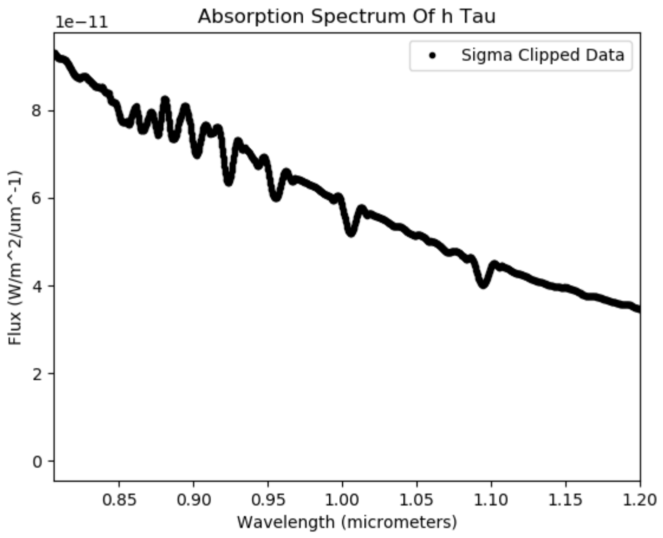

Figure 6: Sigma clipped h Tau spectrum

The absorption lines are much clearer now, and overlaying the two it becomes more apparent.

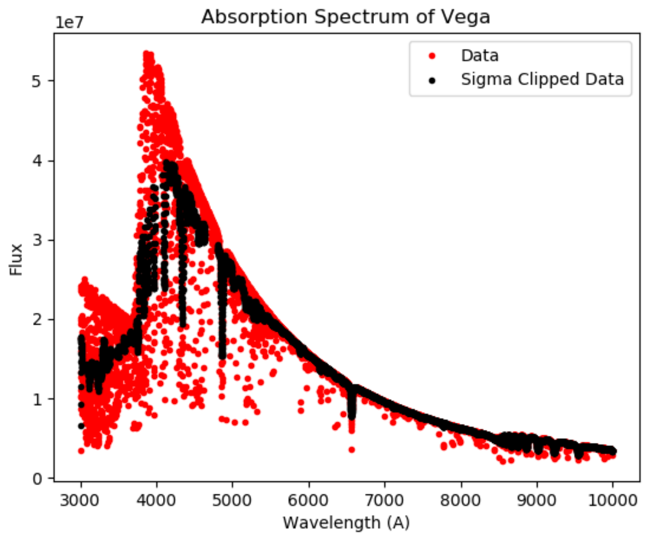

Figure 7: Overlayed Vega spectrum

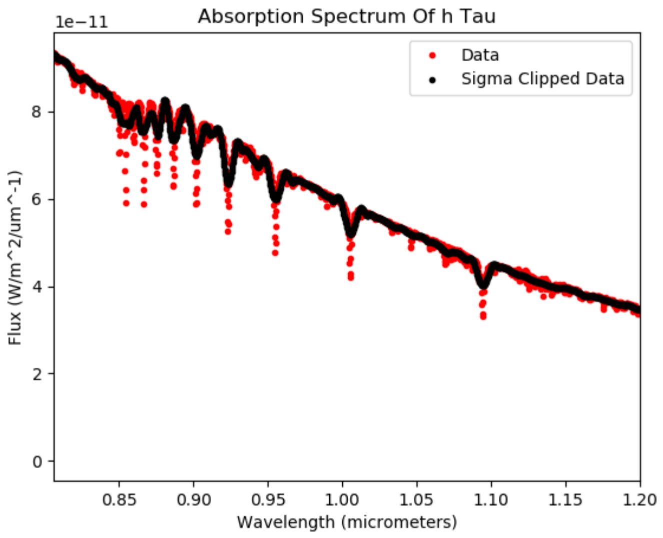

Figure 8: Overlayed h Tau spectrum