

# Assignment 3 - Data estimation

Nishanth Senthil Kumar

Roll No: EE23B049

13th September 2024

## 1 Aim

- The aim of this this assignment is based on trying to estimate various physical parameters using raw data (from the datasets).
- We are given 4 datasets. Initially, we assume that we do not know any of the 4 parameters (which are  $h, c, k_B$ , and  $T$ ), and try to find these parameters through curve fitting.
- Next, we do "partial application", where we assume some of the parameters to be constants, and try to find the remaining parameters through curve fitting.

## 2 NOTE

Please read the instructions at the beginning of the jupyter notebook page, and ensure all requirements are fulfilled before running the code.

## 3 Introduction - What is Curve Fitting

Curve fitting is a statistical technique used to find the best-fitting curve for a set of data points. The goal is to minimize the discrepancies between the observed data and the curve that we model mathematically. One of the most commonly used methods for curve fitting is the *least squares* method, which minimizes the sum of the squared differences between the observed data and the values predicted by the model.

### 3.1 Least Squares Curve Fitting

The least squares method finds the best-fitting curve by minimizing the sum of the squared differences (residuals) between the observed data points and the corresponding predicted values from the model. Let us assume we have a set of  $n$  data points,  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ . We wish to fit a model function  $f(x; \theta)$  to this data, where  $\theta$  represents the model parameters (such as slope and intercept for a linear model).

The least squares objective is to minimize the following sum:

$$S(\theta) = \sum_{i=1}^n (y_i - f(x_i; \theta))^2 \quad (1)$$

Here,  $S(\theta)$  is the sum of squared residuals, where  $y_i$  are the observed values, and  $f(x_i; \theta)$  are the predicted values from the model for the corresponding  $x_i$ . The goal is to adjust the parameters  $\theta$  to minimize  $S(\theta)$ .

## 3.2 Initial Guesses

For nonlinear models (which is what we are dealing with in this assignment), curve fitting algorithms typically rely on iterative methods to find the optimal parameters. An iterative optimization algorithm, starts with an initial guess for the parameters and then adjusts these values to reduce the residuals step by step.

## 3.3 Importance of Initial Guesses

- **Convergence:** Poor initial guesses may lead the algorithm to converge to a local minimum of the error function rather than the global minimum.
- **Speed:** A good initial guess can reduce the number of iterations required to reach the optimal solution, making the fitting process faster.
- **Avoiding Non-Convergence:** In some cases, if the initial guess is too far from the true values, the algorithm may fail to converge.

Therefore, it is often beneficial to choose initial guesses based on prior knowledge of the system or through exploratory data analysis. We will see that a bad initial guess would lead to very bad fits, or no fits at all.

## 4 Part 1 - None of the parameters are known

We will start the assignment by trying to fit the 4 data sets using some curve fitting tool in Python. I have chosen the "curve\_fit" function of the SciPy library.

This function works on the basis of Least Squares Curve Fitting that was explained above i.e uses the Levenberg-Marquardt algorithm

- The first element of the text file is the value of wavelength, and the second element is the value of the function.
- I first store the data from the given text files in NumPy arrays, I read through the file, split each line of the file on the basis of ',', and if each line after splitting has 2 elements, I append it to a list representing the x and y coordinates. If it does not have 2 elements, I drop that line.
- I define the model that I fit my data to, which is the spectral radiance equation, i.e

$$B_{\lambda}(\lambda, T)d\lambda = \frac{2hc^2}{\lambda^5} \frac{1}{e^{\frac{hc}{\lambda k_B T}} - 1} d\lambda$$

- Since I am assuming that none of the parameters are known, I give  $h$ ,  $c$ ,  $k_B$ , and  $T$  as the parameters to fit, along with  $\lambda$  as the x-coordinate.
- I define a list "initial\_guesses" which stores the initial guesses for all 4 parameters. For the first iteration, I have defined the guesses as  $1 \times 10^8$  for  $c$ ,  $1 \times 10^{-23}$  for  $k_B$ ,  $1 \times 10^{-34}$  for  $h$ , and 1000 for the temperature.
- The `curve_fit` function requires the  $x$  and  $y$  arrays as the function arguments, and I have given the initial guesses as an optional argument.
- The `curve_fit` function returns two items: the first one being a list with the optimal values of the parameters, which I have stored in "popt", and the second one being the covariance matrix, which I have stored in "pcov".

- I print out the square root of the diagonal elements of the covariance matrix and the optimal values of each parameter, the square root of the diagonal elements can be used as a measure of error.
- I repeat this for all 4 data sets.

## 5 Observations

- Sometimes for certain initial guess, the function outputs a overflow or divide by zero error. If the initial guess is bad, while going towards the best fit, it may go through some intermediate values which may cause overflows, and the divide by zero error could happen if some parameter becomes too small.
- The diagonal elements of the covariance matrix are the variances of each of the parameters.
- For the d3 dataset, after manually observing the square root of the diagonal elements of covariance matrix and changing the initial guesses according to that, the optimal values seem to be when the initial guess is

```
1 initial_guess = [6e-34, 3e8, 1e-23, 4000] (For h,c,K_b and T)
```

and the optimal values turn out to be

```
1 Errors are [1.11240363e-28 2.67620751e+13 2.21665164e-18 9.44412816e+08]
2
3 Optimal h value: 6.249809406791794e-34,
4 Optimal c value: 300714406.07996106,
5 Optimal kb value: 1.156888465418878e-23,
6 Optimal T value: 4564.98081314197
```

- Repeating the same thing for the other 3 datasets, playing around with the initial guess till the variance was least, we get:
  - For d1 dataset

```
1 initial_guess = [6.4e-34, 3e8,1e-23,3000] (For h,c,K_b and T)
2
3 Errors are [2.65027314e-28 8.59682571e+13 5.46005918e-18
  ↪ 3.78092184e+09]
4 Optimal h value: 5.163928765393971e-34,
5 Optimal c value: 335010023.9714595,
6 Optimal kb value: 8.905901347698295e-24,
7 Optimal T value: 5456.945165542361
```

- For d2 dataset

```
1 initial_guess = [6.4e-34, 3e8,1e-23,3900]
2
3 Errors are [2.31602498e-27 7.76498082e+14 6.03123220e-17
  ↪ 2.54181003e+10]
4
5 Optimal h value: 4.3878120257127e-34, Optimal c value:
  ↪ 296284548.3485389,
```

```

6 Optimal kb value: 9.777024243866955e-24, Optimal T value:
  ↪ 3983.4429147201467

```

– For d4 dataset

```

1 initial_guess = [6.6e-34, 3e8,1.1e-23,3500] (For h,c,K_b and T)
2
3 Errors are [1.27206789e-27 2.84027514e+14 2.04596779e-17
  ↪ 7.52254620e+09]
4
5 Optimal h value: 6.561600095106175e-34,
6 Optimal c value: 293015066.9290548,
7 Optimal kb value: 1.232062756702826e-23,
8 Optimal T value: 4282.948616726832

```

## 6 Graphs

These are the graphs of the curve fits for the four datasets. These plots were done with the help of matplotlib library.

- Fit for D1

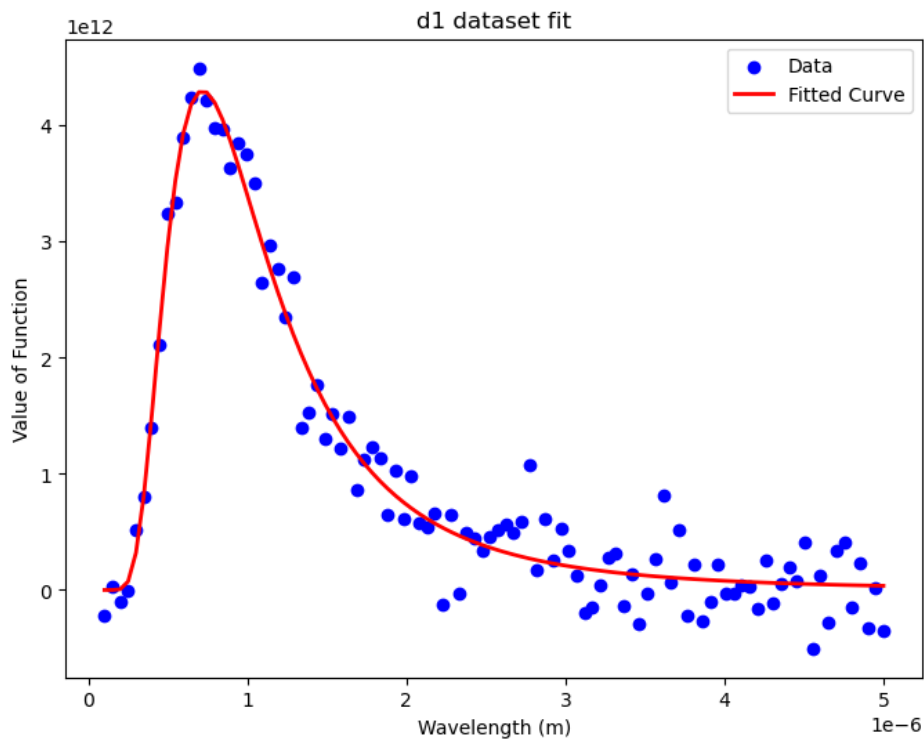


Figure 1: Fit for D1

- Fit for D2

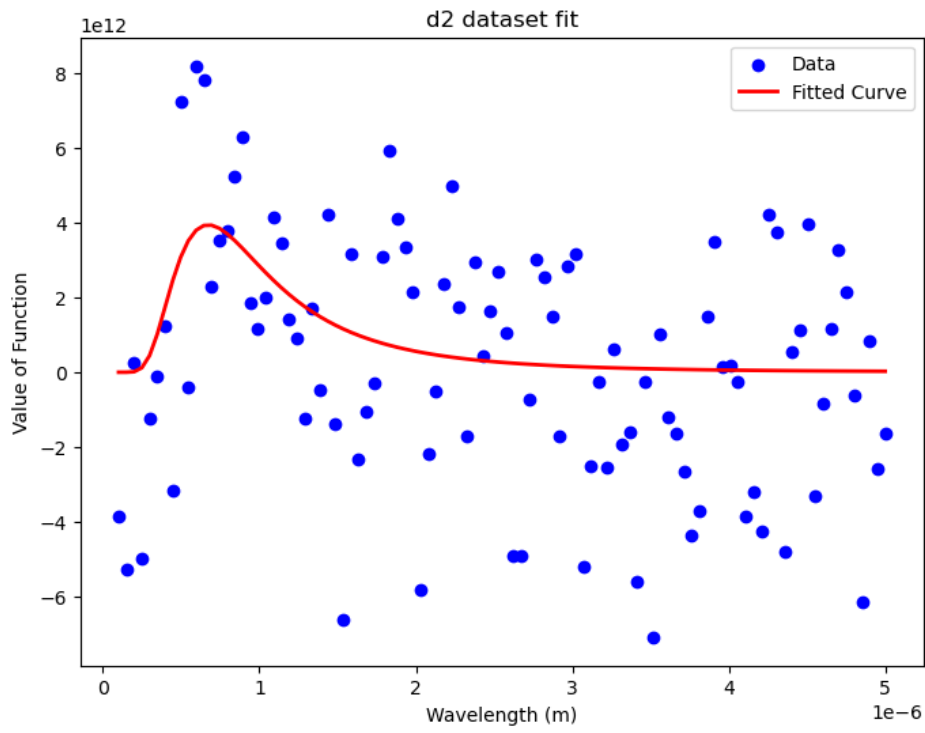


Figure 2: Fit for D2

- Fit for D3

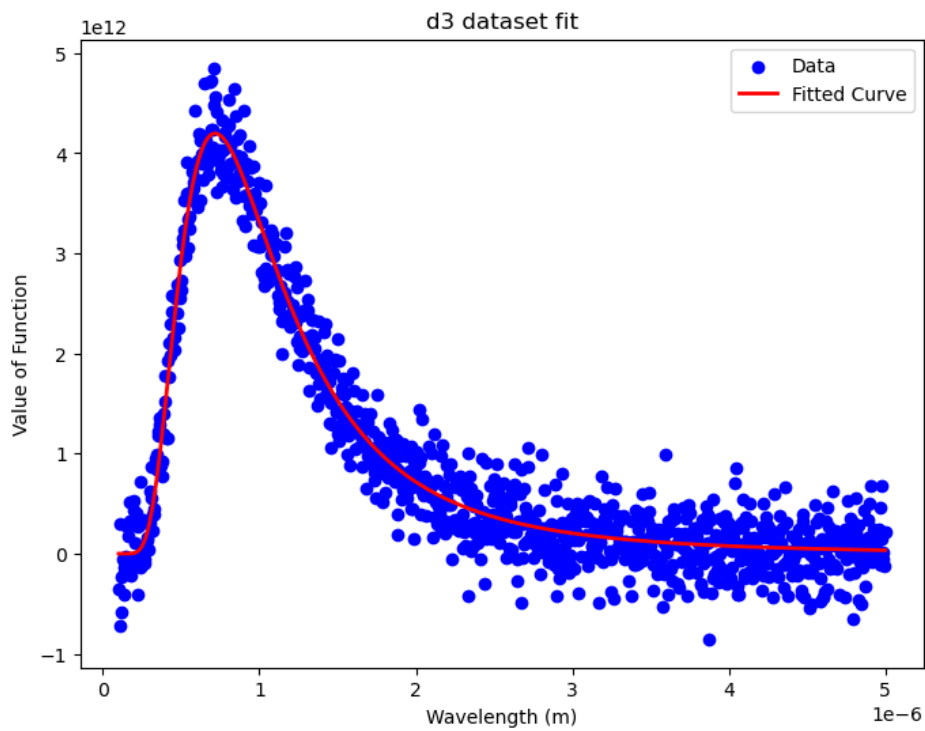


Figure 3: Fit for D3

- Fit for D4

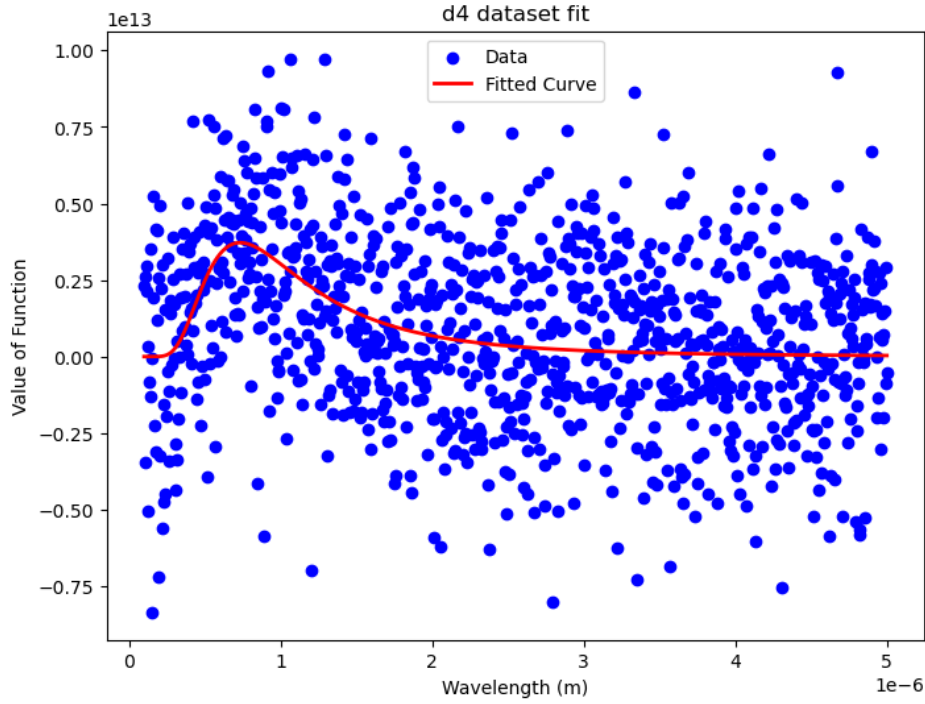


Figure 4: Fit for D4

- We see from the graphs that the d4 and d2 datasets were the most noisy, and d1 and d3 were less noisy, this is reflected in the optimal values of the parameters, as the values are more accurate to the true values in the case of d1 and d3.

## 7 Part 2 - Trying to find T

- In this section, we will keep  $h, c$  and  $k_B$  constant and try to find the value of the temperature at which this data was plotted at.
- The only difference with this part is that the model will have only one parameter apart from the wavelength, so the curve fitting will be done with respect to only one of the variables.
- The values that I have fixed for the constants are:
  - Speed of light,  $c = 3 \times 10^8$  m/s
  - Boltzmann constant,  $k_B = 1.380649 \times 10^{-23}$  J/K
  - Planck's constant,  $h = 6.641 \times 10^{-34}$  J s
- Upon doing the same thing for the 4 datasets, and giving a initial guess of 3000 kelvin for all the four fits, we get the following best fit temperature values.

- For d1 dataset

```
1 Error is [175.79173628]
2 Optimal T value from D2 dataset: [3965.42032515]
```

- For d2 dataset

```
1 Error is [14.29775766]
2 Optimal T value from D1 dataset: [4028.6837984]
```

– For d3 dataset

```
1 Error is [4.79950843]
2 Optimal T value from D3 dataset: [4009.34175797]
```

– For d4 dataset

```
1 Error is [52.46874127]
2 Optimal T value from D4 dataset: [3913.24009218]
```

All these values are in Kelvin

- One nice observation here is that the values of the optimal temperatures do not vary much with the initial guesses, even if the initial guess was way off, it converges to the actual value.
- We can also conclude that d1 was the noisiest dataset and d3 had the least noise.
- As you will see in the next subsection, when only temperature is a parameter, the loss function is a convex function with only one minima, which might be the reason the values turn out to be much better irrespective of the initial guess.

## 8 Extra Part

- I decided to make my own curve fit function to understand how it worked better. I decided to fit the 4th dataset with 3 of the parameters being fixed, and fitting only for temperature, like the second part.
- This was implemented using least squares, similar to the `curve_fit` function, and I just iterate through a set of values of Temperature and find where the loss function has the minimum in that range.
- Loss function, often denoted as  $L$ , is a crucial component in optimization problems and machine learning algorithms. It quantifies the discrepancy between the predicted values and the actual values. For example, in a least squares regression problem, the loss function is typically the sum of squared differences between observed data points and the model's predictions. Mathematically, if  $\mathbf{y}$  represents the vector of actual values and  $\hat{\mathbf{y}}$  represents the vector of predicted values, the loss function can be expressed as:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

where  $n$  is the number of data points. The goal of the optimization process is to minimize this loss function, thereby improving the accuracy of the model.

- I iterated through a range of temperatures from 300 to 5000 with a step size of 0.1, and the output was:

```
1 Optimal T = 3913.2000000008215
```

This is almost the same output that was obtained using the `curve_fit` function.

- Plot of the loss function vs Temperature

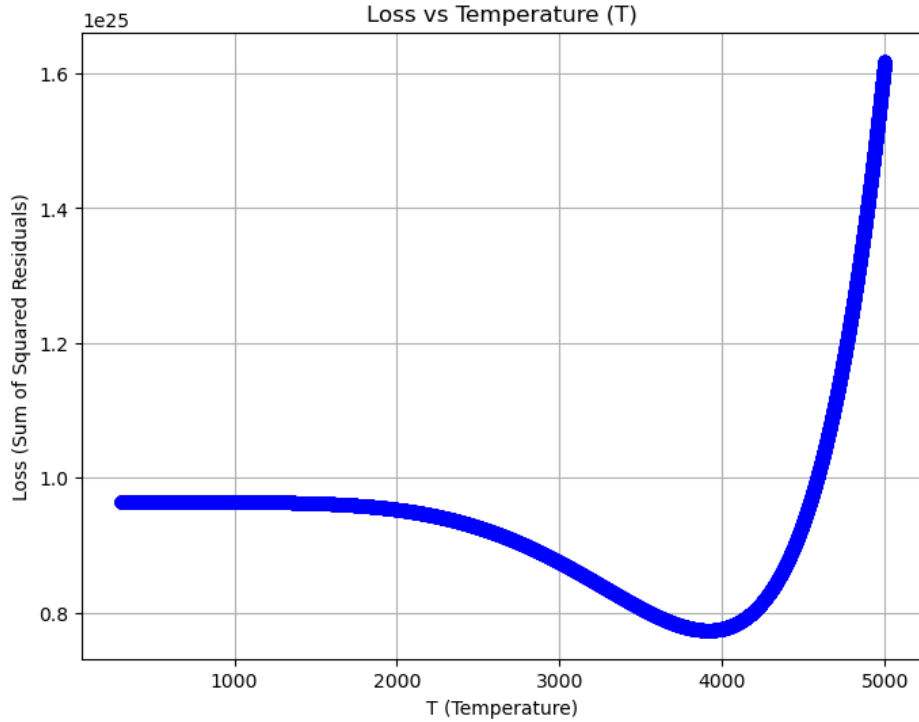


Figure 5: Plot of the loss function vs Temperature

- It can be verified that the temperature values come out to be almost the same for the other 3 datasets.
- However this method has certain flaws, if the range is not proper, it may never have the minima of the function in that range, we can check for that is the minima always turns out to be one of the end points of the range that we specified, and alter that range as and when it is required.
- Calculating the loss function for these many iterations is a very costly process, and this can be optimised by implementing gradient decent, and moving in the direction of 0 gradient. This method will also have issues if there are multiple local minima.

## 9 Important Conclusions

- These are some important conclusions that I have come across only after doing my own curve fit function.
- There were two errors that came up while varying the initial guesses usually, Divide by Zero error and Overflow error.
- Whenever the initial guess lead to T being 0 at some point, Divide by zero error would come up, since the formula had a temperature term in the denominator.
- The reason why Overflow error comes up is much more interesting. A float in python can store values in the range  $-1.79 \times 10^{308}$  to  $1.79 \times 10^{308}$ . If any value ends up outside this range, python stores that number as inf or -inf. Now in the exponential term of the expression, we have the term  $\frac{hc}{\lambda k_B T}$



- Now, take the example of  $T$  being 10 and  $\lambda$  being  $1 \times 10^{-7}$ , the value of the term becomes around 14400, and we are raising that to the power of an exponential. But natural logarithm of  $1.79 \times 10^{308}$  is around 709. As a result, the value will overflow and python will represent it as inf. Python gives an overflow warning when this happens.
- Since it is in the denominator of the expression, it becomes as  $1/\text{inf}$ , which python takes as 0(I verified by trying it out).
- So all the fitted\_y terms for the case where the value of the expression  $\frac{hc}{\lambda k_B T}$  exceeds 709 will be 0.
- So to get more accurate values, it would be wise to drop these points from the dataset.
- These errors are just warnings, and the code wont stop when they come up, but they would lead to inaccurate values, as all these values will give wrong fitted\_y values.
- One more observation is that whenever the value of a float is less than  $1 \times 10^{-323}$ , python automatically takes it as 0, which also may be a cause of DivideByZero error when the guesses are really bad.
- It makes more sense to drop the data point if overflow happens, as it gives an inaccurate fit for that points, I did not implement it here, as the ideal values come when there is not overflow in this case. In a different system, when the ideal answer comes to a set of points where overflow happens, it would be wise to drop that data point when a overflow warning is detected.

## 10 Things to Improve on

- Other curve fitting libraries could be tried out to see how accurate it might be.
- In part 1, the process of finding the best initial guess could have been automated using a loop instead of manually looking at the errors.
- To obtain more accurate fits to noisy datasets, some form of filters could have been applied to the dataset and a curve could have been fit to the denoised dataset.

## 11 References

- SciPy documentation
- Wikipedia page on Planck's constant
- A document on Loss function
- A document on Curve Fitting
- A document on Initial Guesses
- Levenberg-Marquardt algorithm