Assignment 3: Data Estimation
REPORT

EE23B130, Dhruv Prasad

September 15, 2024

1 Introduction

Planck's law describes the spectral radiance of electromagnetic radiation emitted by a black body in thermal equilibrium at a given temperature T. The formula is:

$$S(\lambda, T) = \frac{2hc^2}{\lambda^5} \cdot \frac{1}{\exp\left(\frac{hc}{\lambda k_B T}\right) - 1}$$

Where:

- $S(\lambda, T)$ is the spectral radiance (the power emitted per unit area, per unit solid angle, per unit wavelength).
- λ is the wavelength of the radiation.
- T is the absolute temperature of the black body in kelvins.
- h is Planck's constant (6.626×10^{-34} Js).
- c is the speed of light in a vacuum (3.0×10^8 m/s).
- k_B is the Boltzmann constant (1.38×10^{-23} J/K).

The objective of this assignment is to take the constants as unknown parameters and optimize them, given a dataset of spectral radiance and wavelength values.

2 How to Run

- Download the .ipynb file on a jupyter server or another environment that supports ipympl to show interactive plots in the notebook.
- Run the code blocks cell by cell, or press run all to see the outputs.
- You can then see the markdown and outputs to go through my approach.

- The last code block generates a lot of plots, so I have kept it commented out, as I thought it didn't provide any necessary additional information. If you want to run it it should be uncommented.

Note: I made the .ipynb file in VSCode after pip installing ipython and ipympl, and the jupyter notebook extansion and was able to view the ipympl plots properly.

3 Approach

- I have a class ‘Data‘ to hold all data related to a given file so that I can perform certain operations easily over a list of different files.

  It expects a file containing x and y coordinates data separated by commas.

  It stores these values when initialized with a file.

  Once ‘fit()‘ is called on an instance of ‘Data‘ with a model function and a List of initial guesses for unknown parameters of the model function, it estimates those parameters and stores them.

  We can plot the fitted curve along with the raw data using ‘plot_data_and_fitted_curve()‘. It has parameters to customize the plot.

  Note: ‘plot_data_and_fitted_curve()‘ must be called only once ‘fit()‘ has been called

  Note: The reason behind requiring initial guesses is that if data ranges from highly negative to highly positive powers of 10, the calculations can easily cause overflows. I wasn't able to avoid overflow without a very good initial guess.

```python
# Define a class Data with an initializer
class Data:
    ''' Holds all data related to a given file \n
    '''
    def __init__(self, filename: str):
        '''When initialized with a file name, loads a 2D numpy array with co-ordinates \n
        Creates two numpy arrays to hold x and y values separately \n
        '''
        self.points = np.loadtxt(filename, float, delimiter=',') # Loads coordinates as 2-D numpy array
        self.x_values = self.points[:, 0]  # All rows, first column (x values)
```

```
        self.y_values = self.points[:, 1]  # All rows, second column (y values)

    def fit(self, function: Callable, initial_guesses: List):
        '''Fits curve for a model function and a List containing initial guesses \n
        Returns optimized parameters (Array)\n
        '''
        # This function must take x_values as first argument
        self.model_function = function

        # Using curve_fit() function from scipy.optimize
        self.popt, self.pcov = curve_fit(function, self.x_values, self.y_values, p0=initial_guesses)

        # popt is in the order of next parameters of function
        return self.popt

    def plot_data_and_fitted_curve(self, curve_color: str, data_color: str, x_label: str, y_label: str):
        '''Plots the fitted curve along with the raw data \n
        Returns optimized parameters (Array) and their covariance matrix (2D Array)\n
        '''
        # Begin plotting
        plt.figure()

        # Plot the original data
        plt.plot(self.x_values, self.y_values, label='Data', color=data_color)

        # Plot fitted curve using optimized parameters
        self.new_y_values = self.model_function(self.x_values, *self.popt)
        plt.plot(self.x_values, self.new_y_values, label='Fitted Radiance', color=curve_color)

        # Add labels and a legend
        plt.xlabel(x_label)
        plt.ylabel(y_label)
        plt.legend()

        # Show the plot
        plt.show()
```

- Then, I made a function to find spectral radiance given wavelength, h, c, kb and T, and used it with fit() for the 4 data files given.
- Using very close initial guesses, I was able to get the curve to fit the data pretty well, but the estimated parameters were way off. You can see the plots in the notebook, I have given the optimized parameters below.

File/Parameter	h	c	k_B	T
Initial Guess	6×10^{-34}	3×10^{8}	2×10^{-23}	6×10^{3}
d1.txt	2.05×10^{-33}	1.68×10^{8}	8.28×10^{-24}	1.17×10^{4}
d2.txt	6.92×10^{-34}	2.36×10^{8}	1.36×10^{-23}	3.61×10^{3}
d3.txt	2.73×10^{-33}	1.44×10^{8}	1.56×10^{-23}	7.07×10^{3}
d4.txt	1.90×10^{-33}	1.72×10^{8}	1.35×10^{-23}	6.64×10^{3}

Table 1: Optimized parameters for different input files

- Then, I made partial application functions where only wavelength, h and T or wavelength, c and T were unknowns, and used them with fit for the 4 data files given.
- Despite using much worse initial guesses, I was able to get quite accurate values of h and c. I have given them below. You can see the plots in the notebook.

File/Parameter	h	T
Initial Guess	1×10^{-33}	1×10^{4}
d1.txt	6.43×10^{-33}	3.93×10^{3}
d2.txt	4.27×10^{-34}	2.78×10^{3}
d3.txt	6.27×10^{-33}	3.83×10^{3}
d4.txt	6.25×10^{-33}	3.73×10^{3}

Table 2: h values

File/Parameter	c	T
Initial Guess	1×10^{8}	1×10^{4}
d1.txt	2.95×10^{8}	3.98×10^{3}
d2.txt	2.41×10^{8}	3.46×10^{3}
d3.txt	2.92×10^{8}	3.94×10^{3}
d4.txt	2.91×10^{8}	3.84×10^{3}

Table 3: h values

4 Conclusion

- We cannot predict many parameters at once using noisy data, as the parameters cannot always be separated out. For eg: kb and T in planck's formula appear as a product only.

- By using partial application, we can predict 2 parameters at a time rather reliably.
- Even then, my predicted h and c for d2 showed much greater error than for other files because that file has more noise.