Estimating the Black Body curve

Krutarth Patel ee23b137

14th september 2024

Introduction

This report aims to estimate the various constants in the black body curve using raw data.

The equation for the curve is given by:

$$I(\nu, T) = \frac{2\pi hc^2}{\lambda^5} \cdot \frac{1}{e^{\frac{hc}{\lambda k_B T}} - 1}$$

- $I(\lambda, T)$ is the intensity of radiation at wavelength λ and temperature T,
- h is Planck's constant,
- λ is the wavelength of radiation,
- c is the speed of light in a vacuum,
- k_B is the Boltzmann's constant,
- T is the absolute temperature of the black body.

We are interested in estimating the values of

$$h, c, k_B, T$$

Data

the following samples are provided to us:

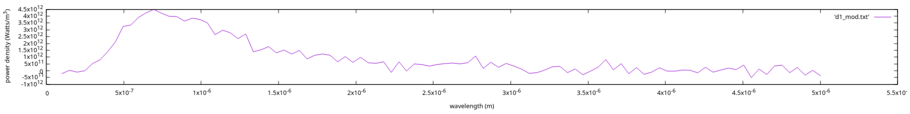


Figure 1: Sample 1

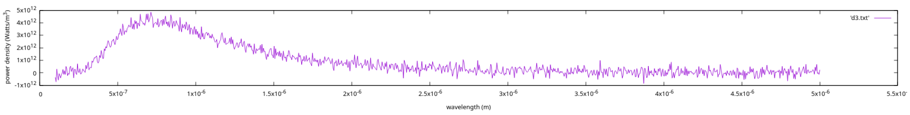


Figure 2: Sample 3

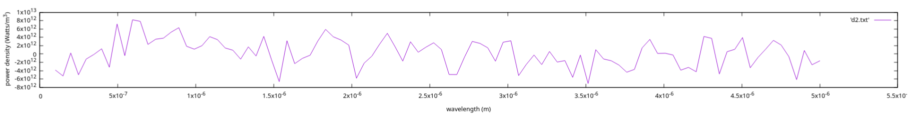


Figure 3: Sample 2

No Initial guess

To begin with we will try to fit the sample points with no prior estimation given. We quickly run into a problem though, trying to fit without an initial guess leads to a float overflow error. This is expected since Scipy assumes an initial value of 1 for all parameters. Assuming

$$h, c, k_B, T = 1$$

and

$$\lambda = 10^{-7}$$

the approximate value of

$$I(\lambda, T) = \frac{2\pi}{10^{-35}} \cdot \frac{1}{e^{10^7} - 1}$$

which for obvious reasons cannot be stored using the precision of a 32-bit floating point number. e^{10^7} is causing the overflow, thus we can **normalize** the samples. Multiplying each input value with 10^6 avoids the overflow error and allows the algorithm to begin iterating.

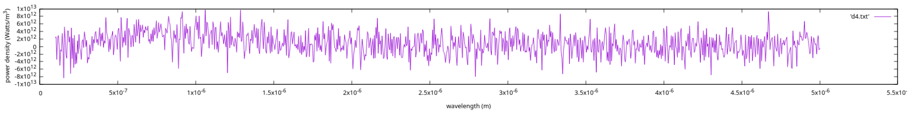


Figure 4: Sample 4

Algorithm

Now is a good time to talk about the tool I am using to fit. my first instinct was to mess with the data using `gnuplot`, it is a command-line plotting tool which is very easy to use. gnuplot has a fitting tool as well, so I tried it first.

Passing the normalized data to the fitting tool in `gnuplot` using the following command:

```
fit f(x) 'd1_normalized.txt' via h,k,c,t
```

But, this does not help much. We get the following fit:

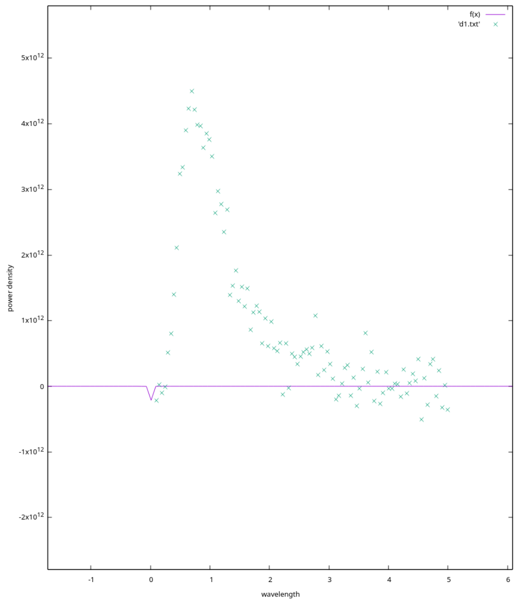


Figure 5: first attempt

Now, to converge the fit properly, I used an approximation of the black body curve:

$$I(\nu, T) = \frac{2\pi hc^2}{\lambda^5} \cdot (e^{\frac{-hc}{\lambda k_B T}} - 1)$$

This gives:

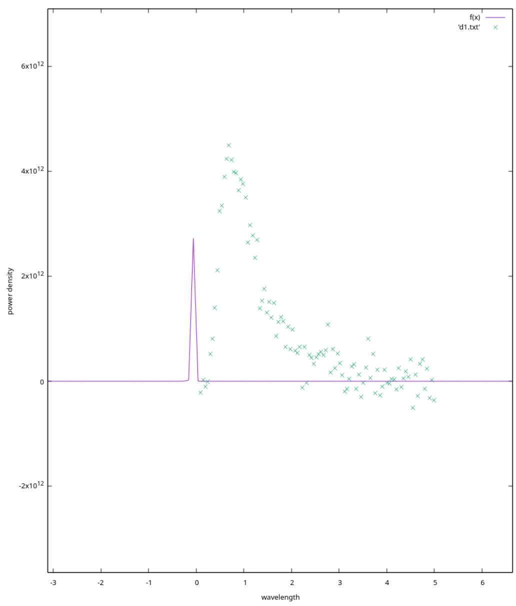


Figure 6: using approximation

Alternating between this approximation and the real formula, I was able to get the following curve:

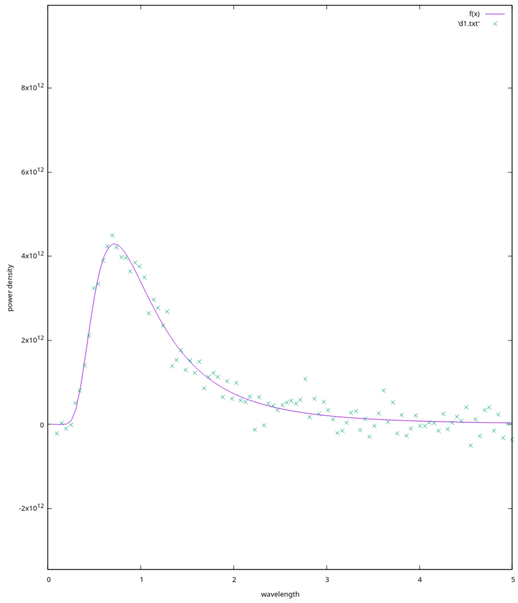


Figure 7: no initial guess(note the values in x axis range from 0-5)

NOTE: This can be accomplished using SciPy as well, The code I have submitted will approximate without a guess and with a guess and show the results.

With Initial guess

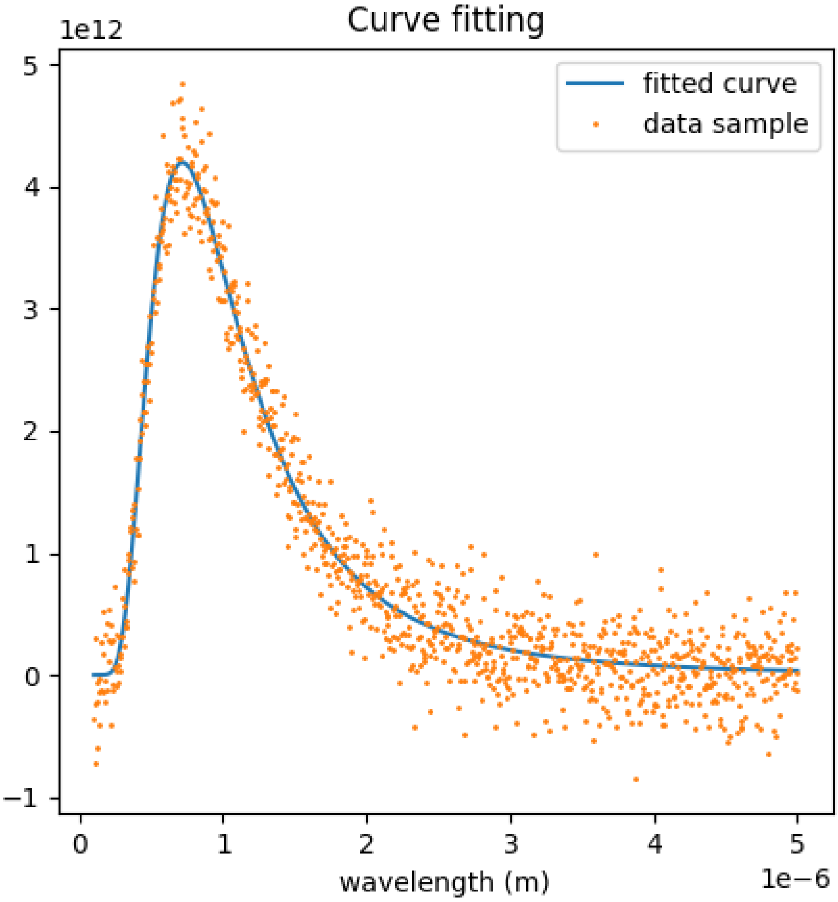
For this section I have used the SciPy libraries `curve_fit` function.

The code for the same looks like this:

```

import scipy
import numpy as np
import matplotlib.pyplot as plt


def f(x, h, c, k, t):
    return (2 * h * c * c / x**5) /
      (np.exp(1 * h * c / (x * k * t)) - 1)


def main():
    xdata = []
    ydata = []
    with open("data/d3.txt", "r") as fin:
        for line in fin.readlines():
            x, y = line.split(",")
            xdata.append(float(x))
            ydata.append(float(y))

    #fitting
    initial_guess = [6.62 * 1e-34, 3 * 1e3, 1.38 * 1e-23, 1]
    params, cov = scipy.optimize.curve_fit(
        f, xdata, ydata, initial_guess, nan_policy="omit"
    )

    #getting prediction
    ypred = []
    for val in xdata:
        ypred.append(
          f(val, params[0], params[1],
            params[2], params[3])
                    )

    #plotting
    plt.title("Curve fitting")
    plt.xlabel("wavelength (m)")
    plt.ylabel("power density (Watts/m^3)")
    plt.plot(xdata, ypred, label="fitted curve")
    plt.plot(
        xdata, ydata, markersize=2, marker=".", linestyle="None", label="data sample"
    )
    plt.legend()
    plt.show()


if __name__ == "__main__":
    main()
```

Since there are three known constants h, c, k_B, we can give the initial guess of any combination of these, starting with:

$$h = 6.62 \times 10^{-34}, c = 3 \times 10^{3}, k_B = 1.38 \times 10^{-23}$$



And the final constants are:

$$h = 2.17146884 * 10^{-31}, c = 1.61328417 * 10^{7}, k_B = -3.53152592 * 10^{-21}, t = -2.78747897 * 10^{2}$$

Using the program

You can reproduce these results with the python file I have attached with my submission. Please keep the following in mind while using the program:

- pass the filename as a positional argument
- As mentioned earlier, I normalize the input data by multiplying with a constant. To get a correct fit, One will have to play around with this normalization constant. For the purpose of this assignment please use the following values:
  - for `d1.txt` use 10^6
  - for `d3.txt` use 10^5

  Thus, running the program would look something like this:

```
$ python3 ee23b137.py d1.txt -normalize 1e6
```