Assignment3CurveFitting

October 2, 2023

1 EE2703: Applied Programming Lab

1.1 Atreya Vedantam, EE22B004

1.2 Dataset 1

Answer: The best linear fit for this data is $y = 2.791124245414918x + 3.8488001014307436$.

1.2.1 How is the M matrix for least squares constructed?

The M matrix is a transformation applied to the parameters of the linear regressor (in this case, labelled p_1 and p_2). The idea is that the M matrix acts as the generative matrix which takes the input x-values of the dataset (labelled `xc` in the code) and produces the (closest fit to) the y-array by acting on a vector of parameters (used to control the transformation). In this problem we want to find the least square linear fit between the datapoints. Hence we assume that the approximate y-values are a linear function of the x-values, i.e,

$$y[i] = p_1 x[i] + p_2$$

for any i^{th} datapoint. This can be written succinctly as

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

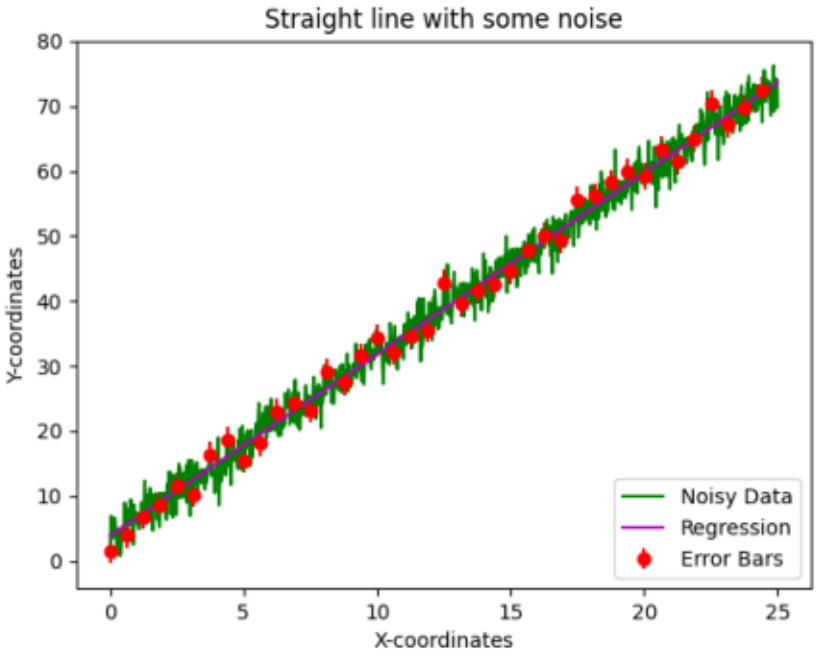
where the equation is $y = Mp$.

1.2.2 Plot of the noisy data with error bars and the estimated line

Given below is the plot of the noisy data with error bars for one in every 25 datapoints with the estimated linear regressor in purple.

[13]:
```
from PIL import Image
import matplotlib.pyplot as plt
plt.grid(False)
plt.axis('off')
img = Image.open('a3_dataset1fig1.png')
plt.imshow(img)
```

[13]:
```
<matplotlib.image.AxesImage at 0x7fd8b75dcca0>
```



1.3 Dataset 2

Answer: The estimated curve (using np.linalg.lstsq) is

$$y = 6.01115293491333\sin(2.5132741228718345x)$$
$$+2.001543820042488\sin(7.5398223686155035x)$$
$$+0.9802390885802212\sin(12.566370614359172x)$$

. The estimated curve (using scipy.optimize.curve_fit) is

$$y = 6.0111529346464\sin(2.5132741228718345x)$$
$$+2.0015438201410234\sin(7.5398223686155035x)$$
$$+0.9802390893231655\sin(12.566370614359172x)$$

.

1.3.1 Estimation of the periodicity of the sine waves (determination of the frequencies)

We are given that the curve is a superposition of three sine waves with frequencies f, 3f, and 5f. Hence finding f is sufficient. A simple good excercise is to visually plot the least square error fit for different frequencies and get a feel for the periodicity of the curve. Once that is done, two extreme frequencies between which the best fit frequency lies were identified (0 and 2 Hertz in the code).

Now iterating through all the frequencies (with a step of 0.1), the one which gave the least error was identified as the best freqency fit.

Alternatively, I could have looked at the signal and written a basic test for checking the periodicity of the signal and derived its fundamental frequency from that.

1.3.2 Construction of the M matrix and equation for least squares

We are explicitly given that this is a superposition of exactly 3 sine waves with frequencies as f, 3f and 5f. Therefore the function is represented by:

$$y = p_1 \sin(2\pi f x) + p_2 \sin(6\pi f x) + p_3 \sin(10\pi f x)$$

, where the factor of 2π converts linear frequency to angular frequency. Hence the M matrix equation will be given by:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} \sin(2\pi f x_1) & \sin(6\pi f x_1) & \sin(10\pi f x_1) \\ \sin(2\pi f x_2) & \sin(6\pi f x_2) & \sin(10\pi f x_2) \\ \sin(2\pi f x_3) & \sin(6\pi f x_3) & \sin(10\pi f x_3) \\ \vdots & \vdots & \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

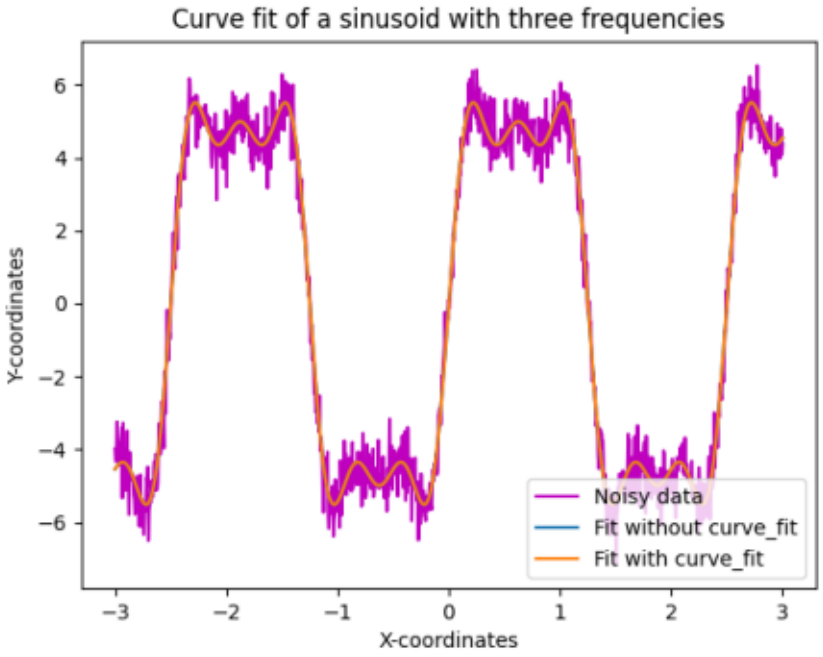
The error is of course given by: $\epsilon^T \epsilon$ where ϵ is the error between the predicted y values and the ones in the dataset ($Mp - g$). This can be minimized by both `numpy.linalg.solve` and `scipy.optimize.curve_fit`. Both give approximately similar answers in this case.

1.3.3 Differences with the solution using `curve_fit`

In this case both approaches yield almost identical results (even when providing `curve_fit` no initial points at all). This is because this function is linear in the parameters and the error chosen is least squares. It results in the optimization problem to be convex and hence far easier to tackle. However the fact that `curve_fit` converges with the default conditions is only a matter of luck and in the third question, more carefully tailored initial conditions need to be chosen for appropriate convergence.

[12]:
```
from PIL import Image
import matplotlib.pyplot as plt
plt.grid(False)
plt.axis('off')
img = Image.open('a3_dataset2fig1.png')
plt.imshow(img)
```

[12]:
```
<matplotlib.image.AxesImage at 0x7fd8b75dc9d0>
```



1.4 Dataset 3: Question 1

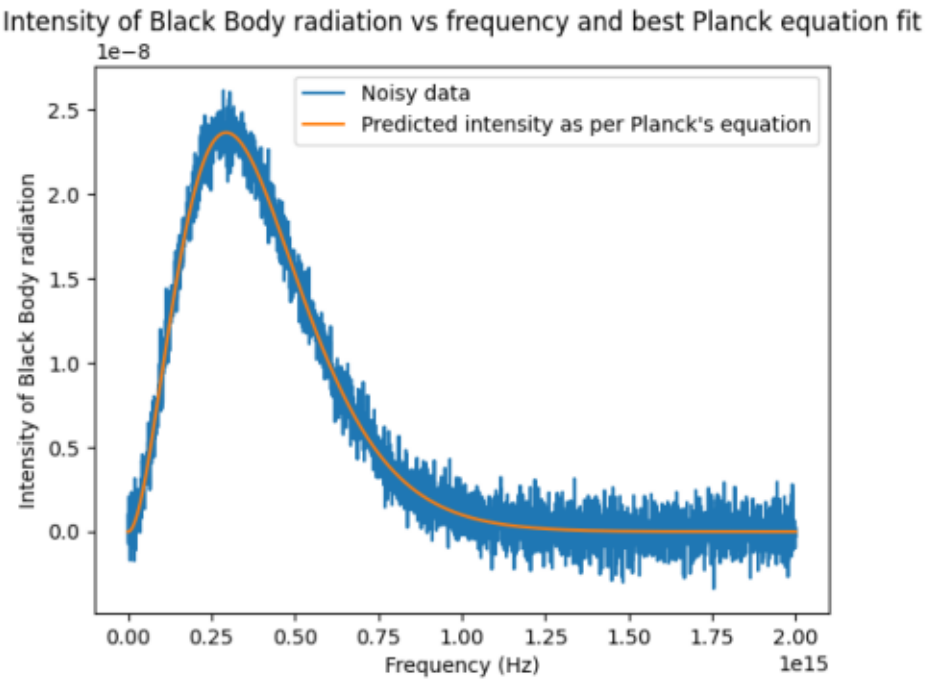
Answer: The temperature at which this data was measured is estimated to be 4997.3419945895985 Kelvin.

1.4.1 Estimation of the temperature of the data (Trial and Error)

At first, I tried running it without supplying any initial conditions (default value for all parameters is 1). It did not converge to any reasonable temperature (it converged to 1 Kelvin) and the fit was pretty bad. Now I removed the curve_fit and start manually playing around with the temperature (modify T and plot the curve). This gave me a good visual sense of how the distribution varied with temperature and I knew approximately what T was a good fit for this data (around 5000 K). Now I supplied the initial condition of 5000 K and it converged to around 4997 K. Now I wanted to test how far away I could go from 4997 and still have curve_fit converge. Turns out that below 500 K it still converges to 4997 K but gives an optimization warning. Not wanting to have that error, I have chosen 500 K as the initial point. Given below is the curve_fit of the intensity of radiation vs frequency.

[14]:
```
from PIL import Image
import matplotlib.pyplot as plt
plt.grid(False)
plt.axis('off')
img = Image.open('a3_dataset3fig1.png')
plt.imshow(img)
```

[14]: <matplotlib.image.AxesImage at 0x7fd8b74fea00>



1.5 Dataset 3: Question 2

Answer: The estimated parameters are: T = 5048.22493614041 K h = 6.752677296893756e-34 Js kb = 1.3941465688609095e-23 J/K c = 303,198,038.87774557 m/s

1.5.1 Estimation of the Planck's constant, Boltzmann constant, speed of light and the Temperature

The only difference between the previous excercise and this one is that there are no initial conditions provided in this question. Here all the constants are taken to be 'learnable' parameters. Hence there is a lot of volatility and the curve_fit function gives us very error-prone results if the initial conditions are not chosen well.

1.5.2 Quality of the Results and Avenues of Improvement

Since plotting the appropriate curve whilst modifying four different parameters is a time-consuming work, I instead have chosen to enter near-accurate starting points for the parameters and tried to see how accurate the results were. My plan was to then gradually move the initial conditions away from the near-accurate ones and see if the final values agreed well with the true values. Unfortunately, it turned out that even for near-accurate values, curve_fit doesn't give us great final answers. Notice the answer above for T (5048 K when it is supposed to be 4997 K). One might think that this error

is not too big but keep in mind that the initial conditions supplied are T = 4997, h = 6.626e-34, kb = 1.38e-23, c = 3e8.

I feel that some avenues of improvement can be found in restricting the values of some constants to a particular range. This might help the optimizer to avoid sharp gradients or pitfalls which can severely affect other parameters.

[15]:
```
from PIL import Image
import matplotlib.pyplot as plt
plt.grid(False)
plt.axis('off')
img = Image.open('a3_dataset3fig2.png')
plt.imshow(img)
```

[15]:
```
<matplotlib.image.AxesImage at 0x7fd8c0655cd0>
```