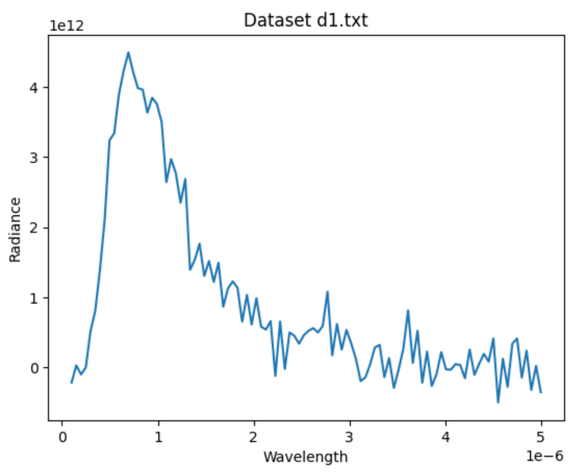
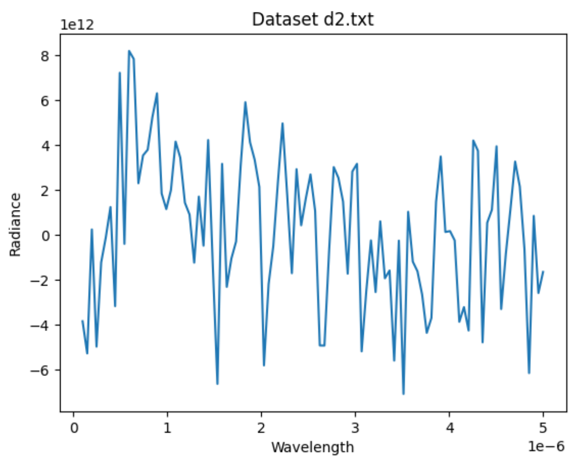
Experiment-3

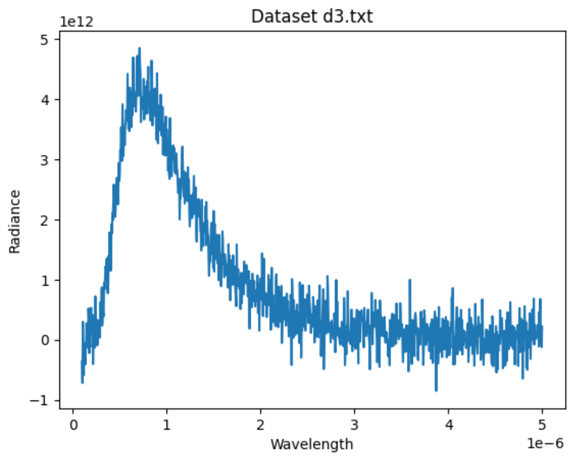
Deenabandhan N
EE23B021

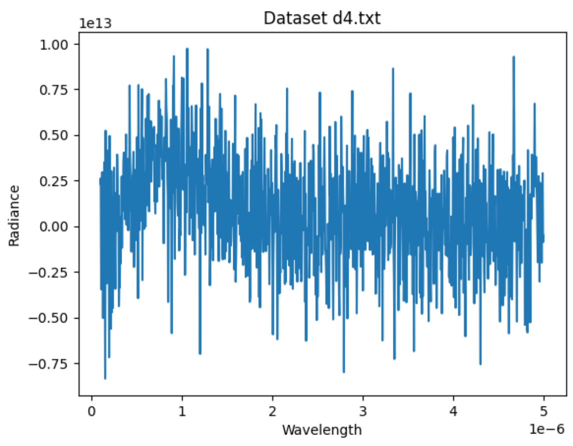
Introduction :

- In this assignment we are given with four datsets that satisfy the Planck's formula experiment.
- The catch is the datsets have some noise of varying magnitude added to it.
- For example **d1.txt** and **d3.txt** have noise of lower magnitude than that of **d2.txt** and **d4.txt**









- We can infer that the noise level in dataset d2.txt and d4.txt is so large.
- The motto of the experiment is to fit the Planck's formula

$$\text{Radiance} = \frac{\text{hc}^2}{\lambda^5}\frac{1}{(e^{\frac{hc}{k_{\text{B}}T}} - 1)}$$

and determine the parameters **h : Planck's constant** , **c : Velocity of light** , k_B **: Boltzmann constant** nad **T : Temperature at which the experiment is conducted**

Ambiguity and issues with the problem statement :

- In this assignment , we are asked to fit the curve with four unknown parameters but in the formula there are only two parameters which can be varied i.e

$$\text{hc}^2 \text{ and } \frac{hc}{k_B T}$$

- So while curve-fitting , we can vary only these two parameters and not the individual parameters such as h , c, k_B and T.
- The most important difficulty faced while fitting the curve using the module **scipy.optimize.curve_fit()** is the overflow issue.

```
RuntimeWarning: overflow encountered in
exp in
return((2*h*(c**2))/((_lambda**5)*(np.exp(val)-1)))
```

- This overflow error is due to the exponential term in the Planck's formula.
- While fitting the curve , the curve_fit() function performs gradient descent thereby taking some initial state.
- As the original values are at extreme magnitudes like $h \approx 10^{-34}$ and $k_B \approx 10^{-23}$ and $c \approx 10^9$ to which the magnitude of initial values do not match.
- So the curve_fit() function raises overflow error.

Ways to solve the assignment :

- First we need to know the optimal initial values where the overflow error won't happen and gradient descent will be useful.
- So I solved the assignment in the reverse manner i.e Solved the partial findings first and then discussed the ambiguity about the first part of the assignment.
- I decided to do this way to find the optimal initial guesses and to discuss the ambiguity clearly.

Second part :

- In this part , we have hard-coded the values of three parameters and left the remaining parameter to vary and fit the curve.
- Even in this partial fit , we may encounter the overflow error.To rectify this , I have taken a range of the varying parameters and found the range where it gives reasonable value and where error attains a minimum value.
- I started this code by finding T parameter as it is completely unknown to us.

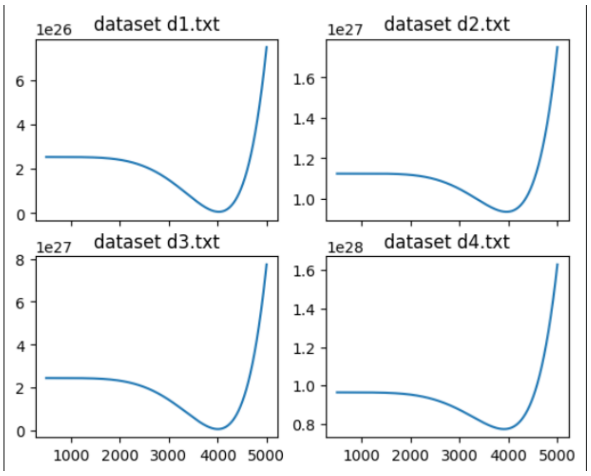
Partial fit using Temperature :

- I have defined a function **fn_T()** which takes wavelength and temperature as inputs and gives the radiance as output.

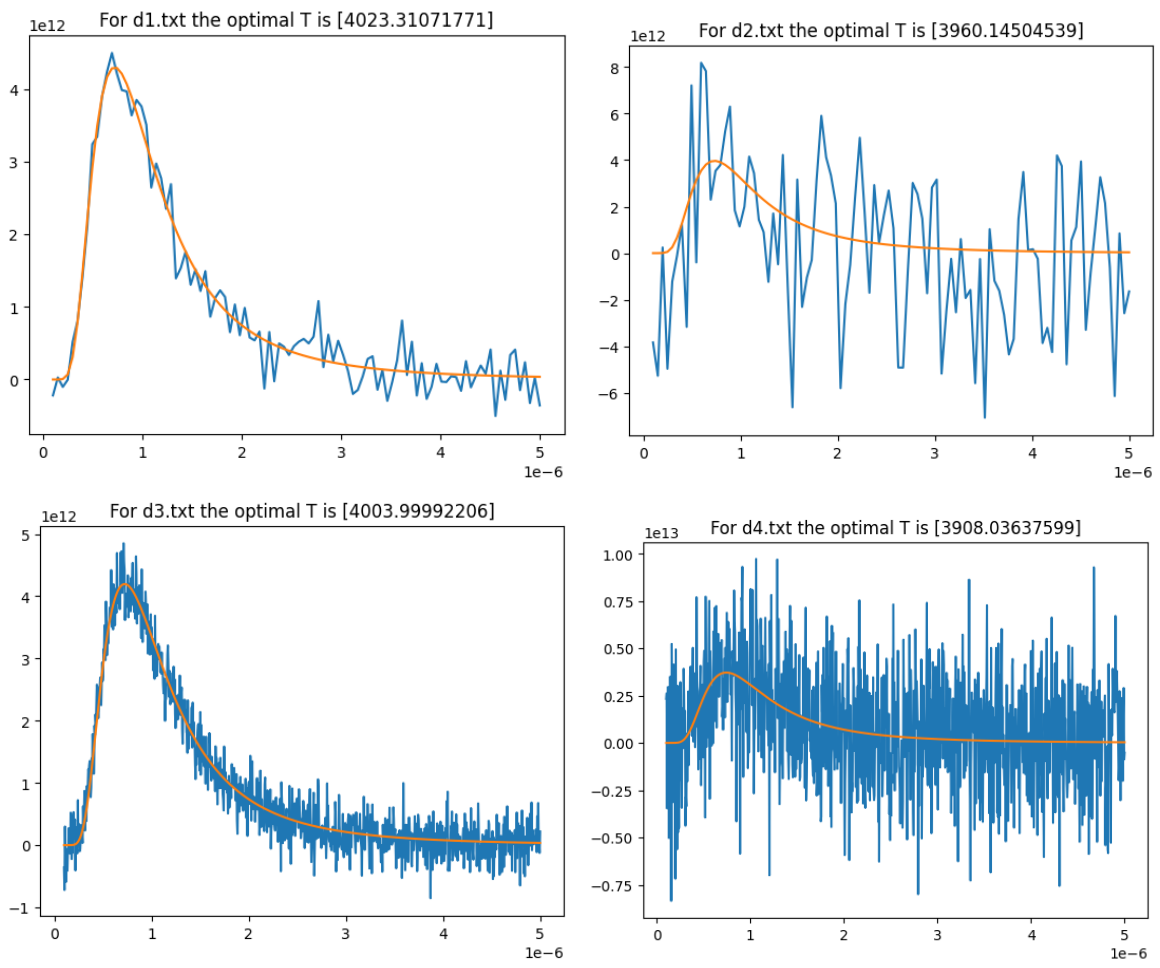
```
def fn_T(_lambda,T):
    c=3e8 #Defining the c value
    k=1.38e-23 #Defining the k_B value
    h=6.626e-34 #Defining the h value

    val=(h*c)
    val=val/(_lambda*k*T)
```

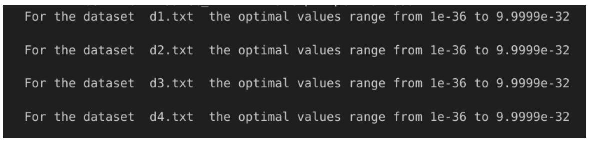
```
        val=np.array(val)

        return((2*h*(c**2))/((_lambda**5)*(np.exp(val)-1)))
```

– I have taken a range of T values (500,5000) to find the optimal temperature value to give as initial guess.
– I have plotted the cost function for varying T and found that minimum occurs somewhere near 4000K.



– So , I decided to give 4000 as a initial guess for the partial fit of datasets.
– The partial fit for the given initil fit comes out to be as follows...





```
For the dataset  d1.txt  the optimal values range from 1e-36 to 9.9999e-32

For the dataset  d2.txt  the optimal values range from 1e-36 to 9.9999e-32

For the dataset  d3.txt  the optimal values range from 1e-36 to 9.9999e-32

For the dataset  d4.txt  the optimal values range from 1e-36 to 9.9999e-32
```

Partial fit by varying h:

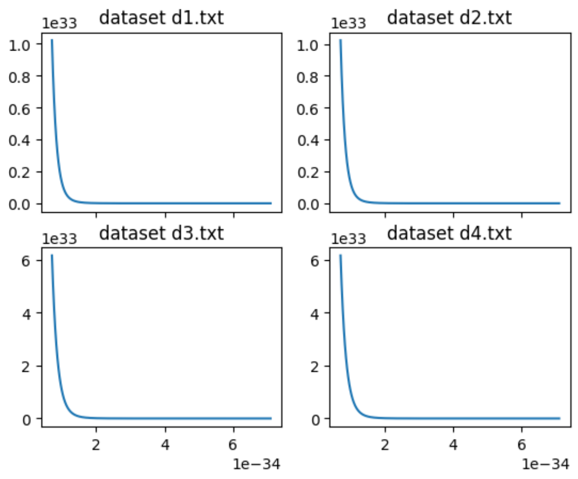
- I have defined a function **fn_h()** which takes wavelength and h as inputs and gives the radiance as output.
- Finding the optimal initial guess here is more subtle as there are cases where we can encounter overflow error.
- So, I have defined a

  ```
  try: #Statement
  except: #Statememnt
  ```

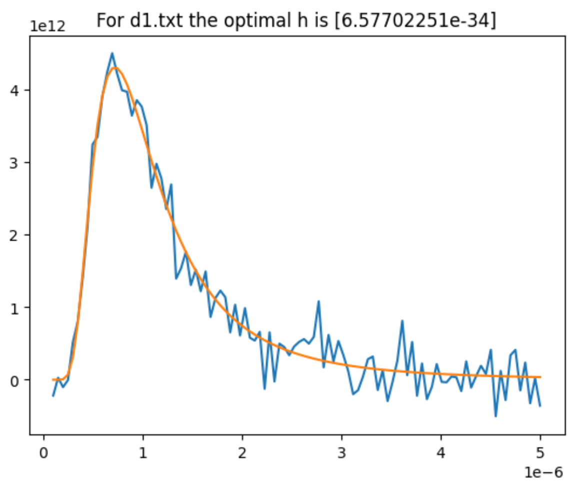
  to find out the places where it raises error and neglect those as our initial guesses.
- So when the code encounters overflow error , it will return [-1e10] which is a placeholder for finding the error guesses.

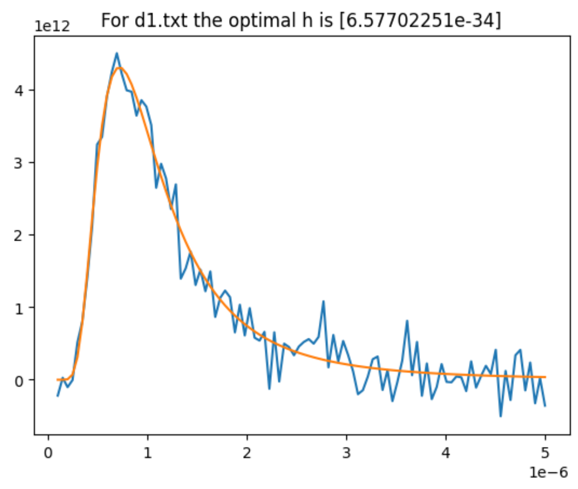
```
def fn_h(_lambda,h):
    c=3e8
    k=1.38e-23
    T=4000

    val=(h*c)
    val=val/(_lambda*k*T)
    val=np.array(val)

    try:
        return((2*h*(c**2))/((_lambda**5)*(np.exp(val)-1)))

    except RuntimeError or RuntimeWarning as e:
        return ([-1e10])
```

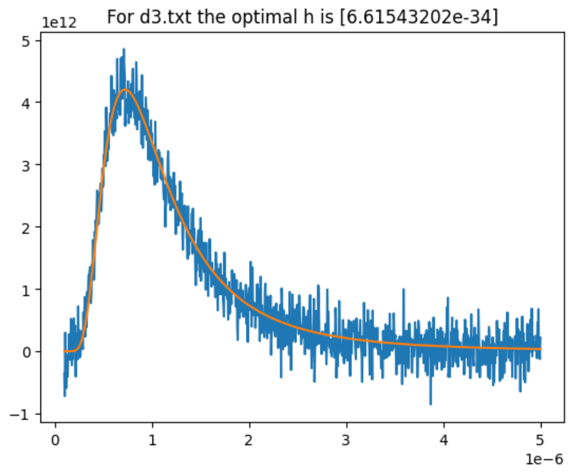
- The range of values for which there is no overflow error is ,
- I have taken a range of h values (1e-35,1e-32) to find the optimal h value to be given as initial guess.
- I have plotted the cost function for varying T and found that minimum occurs somewhere in the range (2e-34,10e-34) .

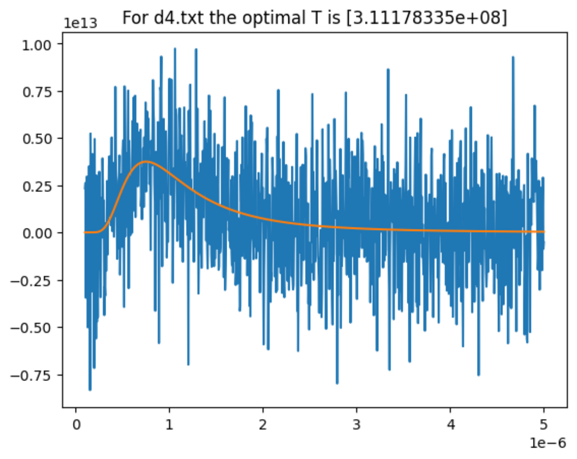


- So , I decided to give 3e-34 as a initial guess for the partial fit of datasets.
- The partial fit for the given initial fit comes out to be as follows...









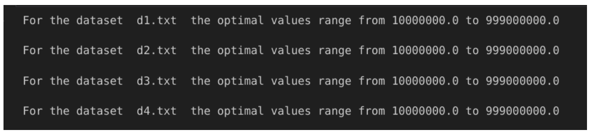
Partial fit by varying c :

- I have defined a function **fn_c()** which takes wavelength and c as inputs and gives the radiance as output.
- Finding the optimal initial guess here is more subtle as there are cases where we can encounter overflow error.
- So, I have defined a

  ```
  try: #Statement
  except: #Statememnt
  ```

  to find out the places where it raises error and neglect those as our initial guesses.
- So when the code encounters overflow error , it will return [-1e10] which is a placeholder for finding the error guesses.

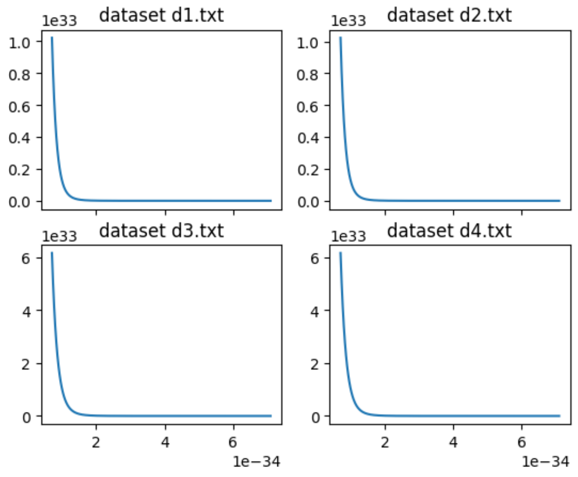
```
def fn_c(_lambda,c):
    h=6.626e-34
    k=1.38e-23
    T=4000

    val=(h*c)
    val=val/(_lambda*k*T)
    val=np.array(val)

    try:
        return((2*h*(c**2))/((_lambda**5)*(np.exp(val)-1)))

    except RuntimeError or RuntimeWarning as e:
```



```
For the dataset  d1.txt  the optimal values range from 10000000.0 to 999000000.0

For the dataset  d2.txt  the optimal values range from 10000000.0 to 999000000.0

For the dataset  d3.txt  the optimal values range from 10000000.0 to 999000000.0

For the dataset  d4.txt  the optimal values range from 10000000.0 to 999000000.0
```

```
            return ([-1e10])
```

- The range of values for which there is no overflow error is ,
- I have taken a range of h values (0.9e7,4e8) to find the optimal h value to be given as initial guess.
- I have plotted the cost function for varying T and found that minimum occurs somewhere in the range (0.9e8,10e8) .



- So , I decided to give 1e8 as a initial guess for the partial fit of datasets.
- The partial fit for the given initial fit comes out to be as follows...

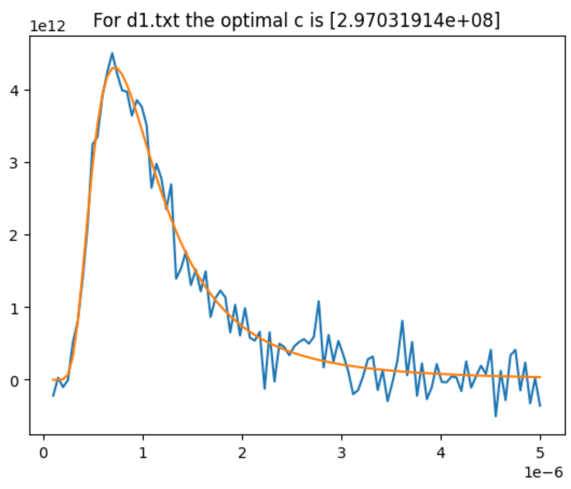
Partial fit by varying k:

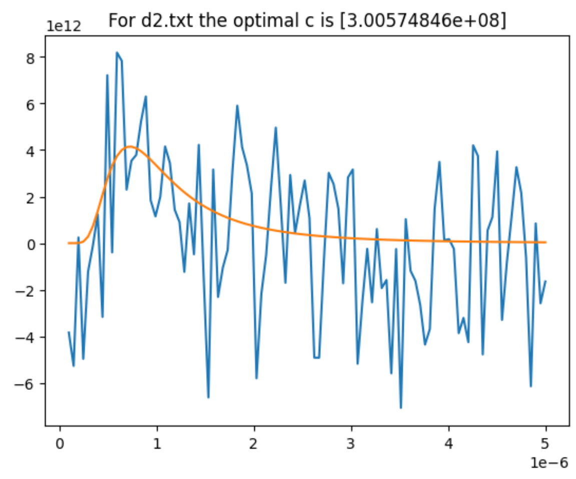
- I have defined a function **fn_k()** which takes wavelength and k as inputs and gives the radiance as output.
- Finding the optimal initial guess here is more subtle as there are cases where we can encounter overflow error.
- So, I have defined a

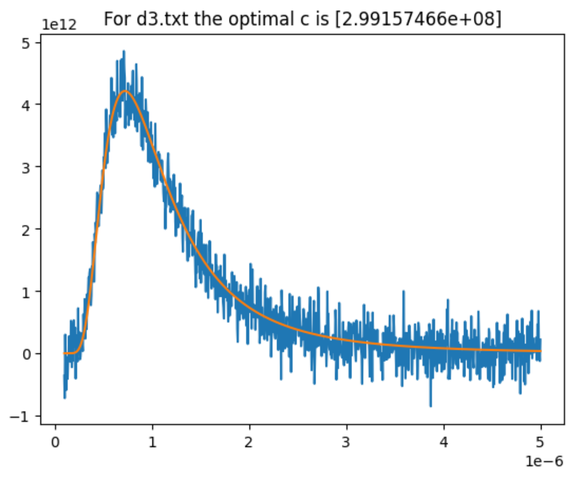
```
    try: #Statement
    except: #Statememnt
```

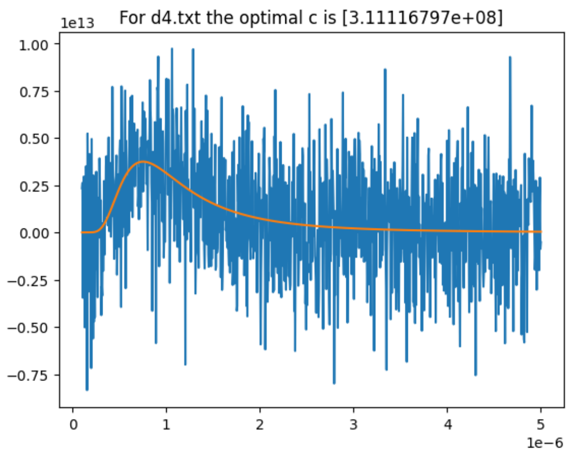
  to find out the places where it raises error and neglect those as our initial guesses.
- So when the code encounters overflow error , it will return [-1e10] which is a placeholder for finding the error guesses.

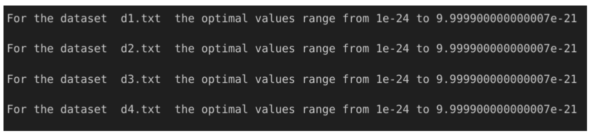
```
def fn_k(_lambda,k):
    c=3e8
    h=6.626e-34
    T=4000

    val=(h*c)
    val=val/(_lambda*k*T)
    val=np.array(val)

    try:
```







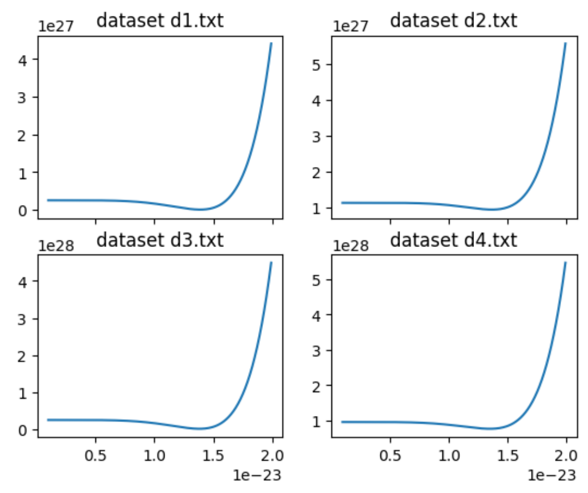




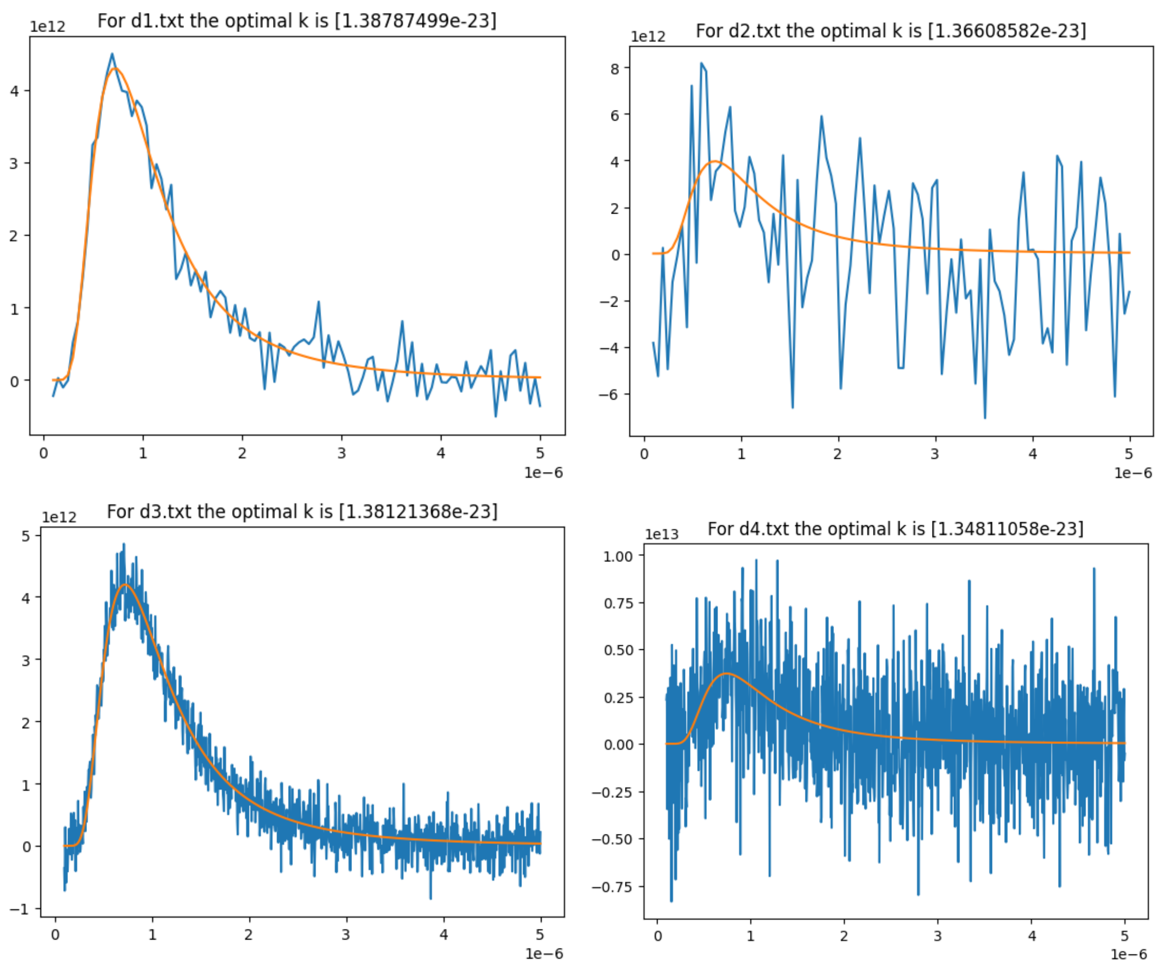
```
For the dataset  d1.txt  the optimal values range from 1e-24 to 9.999900000000007e-21

For the dataset  d2.txt  the optimal values range from 1e-24 to 9.999900000000007e-21

For the dataset  d3.txt  the optimal values range from 1e-24 to 9.999900000000007e-21

For the dataset  d4.txt  the optimal values range from 1e-24 to 9.999900000000007e-21
```

```
        return((2*h*(c**2))/((_lambda**5)*(np.exp(val)-1)))

    except RuntimeError or RuntimeWarning as e:
        return ([-1e10])
```

- The range of values for which there is no overflow error is ,
- I have taken a range of h values (1e-24,1e-20) to find the optimal k value to be given as initial guess.
- I have plotted the cost function for varying T and found that minimum occurs somewhere in the range (1e-23,1.5e-23) .



- So , I decided to give 1.5e-23 as a initial guess for the partial fit of datasets.
- The partial fit for the given initial fit comes out to be as follows...



First part :

- In this part of the assignment , we are trying to fit the dataset with the planck's formula by varying all four parameters.

- As explained already, the issue here is the planck's formula has only two variables in general i.e Planck's formula can be written as

$$\text{Radiance} = \frac{\mathbf{a}}{\lambda^5}\frac{1}{(e^{\mathbf{b}} - 1)}$$

  where $\mathbf{a}$ = hc and $\mathbf{b} = \frac{hc}{k_B T}$

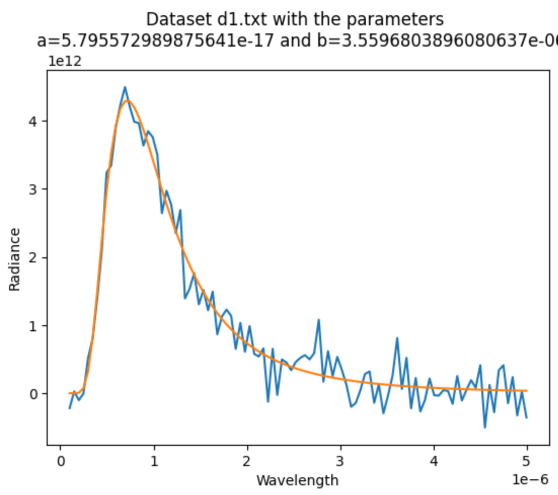
- In this report , I have tried both the ways of fitting the curve ,
  - Giving four parameters
  - Giving two parameters

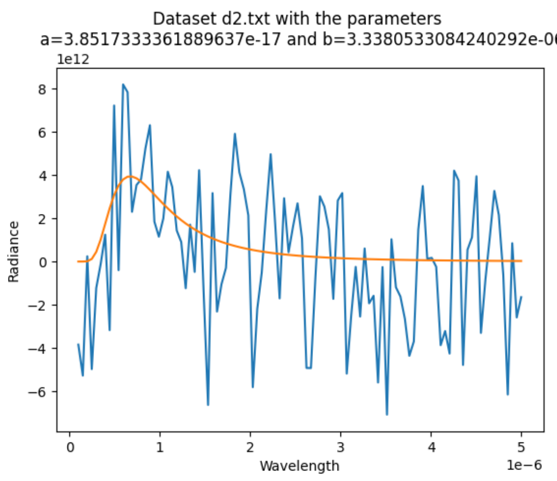
Giving two parameters :

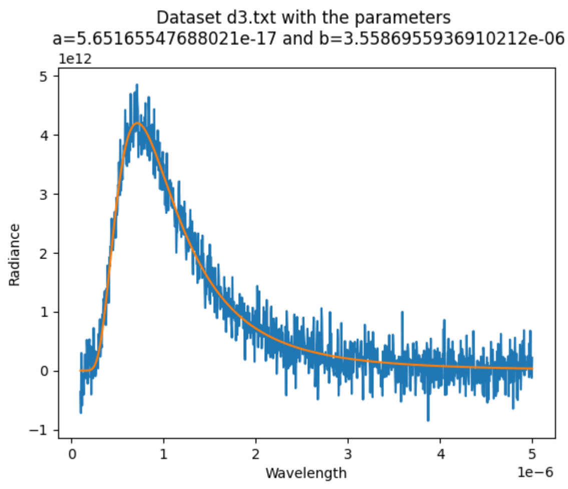
- I have tried to use only two parameters to fit the curve with the given dataset by the following function.

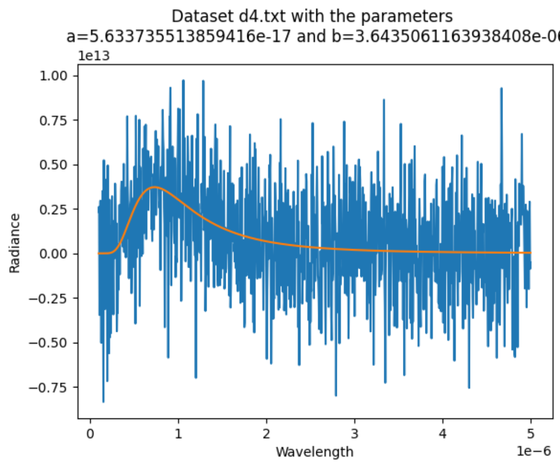
```
def fn2(_lambda,a,b):
     a=np.array(a*2)
     b=np.array(b)
     try:
          return((a)/((_lambda**5)*(np.exp((b/_lambda))-1)))
     except RuntimeError or RuntimeWarning as e:
          print(_lambda)
```

- We have observed that even here there is a chance of overflow error and to avoid that we have to include the initial guess value.
- We will give this value from our estimated value which is found in second part.
- So the initial guess here would be [1e-17,1e-6]. On fitting the curve by varying the two parameters we get the following results ,









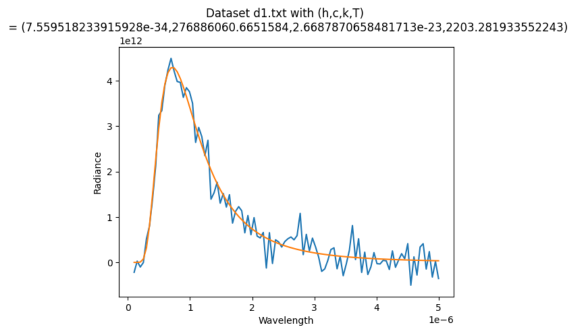
- Using the two parameter function , we have fit the curve such that the actual h,c,k,T of the dataset will yield a,b values close to that of the current result.
- We will now use the four parameter function to fit the curve with the datasets and check if the values match each other.

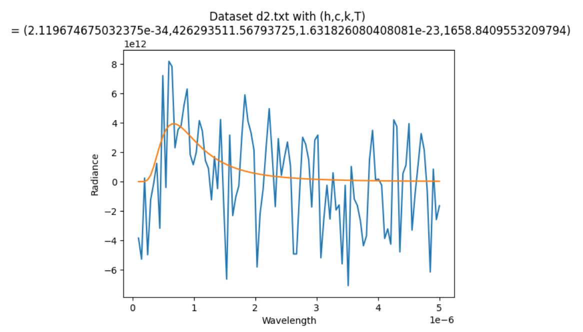
Fitting with four parameters

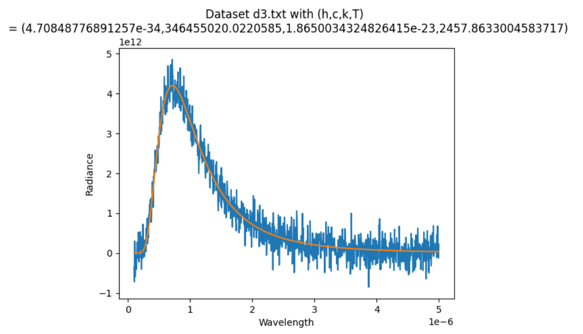
- I have used four parameters to fit the curve with the given dataset by the following function.

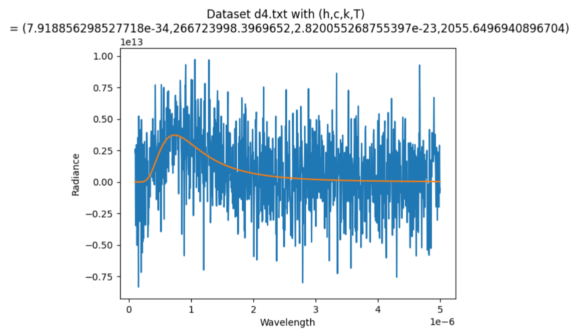
```
def fn4(_lambda,h,c,k,T):

    val=(h*c)
    val=val/(_lambda*k*T)

    val=np.array(val)

    return((2*h*(c**2))/((_lambda**5)*(np.exp(val)-1)))
```

- To avoid overflow error , we will give some reasonable initial guess.
- We will give this value from our estimated value which is found in second part.
- So the initial guess here would be (h,c,k,T)= [1e-34,1e8,1e-23,1e3].
- On fitting the curve by varying the two parameters we get the following results ,









- We can infer that the values are not accurate as it has to vary four parameters and account for the noisy data too.
- Let us now see the interconnection between using two and four parameters.

Interconnection of two methods

- Let us take a dataset **d3.txt** and fit it with same initial conditions of a and b .
- Let us fit the curve with two parameters and get a_fit and b_fit.
- For the 4 parameter curve fitting let the four initial parameters can vary such that their a and b combination is constant.
- On taking the h values in the range (1e-34,1e-33) and k values in range (1e-23,1e-22) and finding the curve_fit() for these 4 parameters , we will get **h_fit , c_fit , k_fit and T_fit**.
- On finding their a_fit1 and b_fit1 equivalent let us find the percent error corresponding to the a_fit - a_fit1 and b_fit - b_fit1, we get the following result.

```
The largest percentage error of a is 0.0006043507575537266
The largest percentage error of b is 0.00035718788256344737
```

- From the above result , we infer that taking two parameters is same as taking four parameters over a range of numbers.

References :

- sci-py documentation
- Curve-fitting techniques

- I would like to thank my friends :
  - Koushal EE23B031
  - Nishanth EE23B049

for giving me ideas on proceeding through the assignment.