Assignment 3 - Fitting Data to Models

Aravint Annamalai, EE18B125

February 11, 2020

1 Abstract

In this assignment, I am dealing with file handling in Python. Graph plotting is done in Python which is an extremely useful tool in visualising the data, especially when the data is huge with many rows and columns, as is commonly encountered in various day-to-day applications when Machine Learning is used.We are also going to use different kinds of plotting like the simple linegraph, contour plot, scatter plotting. We also see the effect of noise in the fitting process, and how noise impacts the calculation of certain parameters. Also, we will be dealing with mean square error in order to get values of some parameters and also to estimate how perfectly our model fits the data.

2 Introduction

In this assignment, we will be dealing with a class of functions called **Bessel function**. Bessel functions are solutions to the differential equation

$$x^2\frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 + a^2)y = 0$$

We are going to add different amounts of noise to the Bessel function. Here, the noise is assumed to be a normal distribution with mean 0 and certain variance specified by us.

3 Task 1

Here, we are given a Python file which first populates the file `fitting.dat`. The first column contains a vector **t** whose elements are uniformly distributed between 0 and 10 with step size of 0.01. Next the vector **f(t)** is generated which is related to vector **t** as follows:

$$f(t) = 1.05J_2(t) - 0.105t \tag{1}$$

9 copies of this is created and noise of different standard deviations are added to each one of them. The graph between t and the function with noise is plotted. These values are written onto the file `fitting.dat`.

4 Task 2 and 3

Using `numpy.loadtxt()` method, the file `fitting.dat` is loaded. For every column(apart from the first one) in the file, a graph is plotted between itself and the **t** vector stored in the first column. Legends are also marked appropriately.

5 Task 4

A function `g(t, A, B)` is defined as follows:

```
def g(t, A, B):
    x = (A * sp.jn(2,t)) + (B * t)
    return x
```

This basically implements the function f(t) but with a general value of coefficients. Now a graph is plotted with t on the x axis and g(t, 1.05, -0.105) on the y-axis.

6 Task 5

Here, I have plotted the graph which was done in Task 4. The second column of the file `fitting.dat` is plotted in the form of errorbars. Since the values of t are really close together, plotting all the errorbars will make the graphs unreadable and it will be of no use. So, we plot every fifth errorbar so that the errorbars will be neatly spaced out. This can be done using the following code:

```
data = file[:, 1]
t = file[:, 0]
errorbar(t[::5], data[::5], 1, fmt = 'ro')
```

Since we have the actual curve and the curve with noise in the form of errorbars, we can see the value of noise and how noise behaves like.

7 Task 6

We have earlier used the function `g(t, A, B)` to calculate the exact value of f(t) for a given value t. Here, we are going to do the same thing, but use a different method involving matrix operations.

7.1 Function

The function `g1(t, A, B)` is defined here. First it populates the matrix M which has 101 rows and 2 columns. Second column contains the **t** array and the first column contains the Bessel function J(t) . These two are created as separate arrays `col1` and `col2`, which can then be merged into a matrix M by the following code:

```
M = c_[col1, col2]
```

A column array(of order 2) **p** is initialised with the values A and B passed as arguments to the function. Our final result which is to be returned is the matrix product of M and p.

7.2 Function call and equality checking

We define two constants `A0 = 1.05` and `B0 = -0.105` and call the functions `g(t, A0, B0)` which was defined in Task 4 and `g1(t,A0,B0)` which we defined just now. Now we have to ckheck whether the returned matrices are equal or not. For that, we use the function `np.all()` and pass the two arrays as arguments to the function. If the matrices are equal, it retuns `True`. Since the two matrices here are equal, `True` is returned in this case.

8 Task 7

Here, we have to find the mean squared error of the actual vs predicted values for different values of A and B. Two lists A and B are initialised, with A containing values from 0 to 2 in steps of 0.1 and B containing values from -0.2 to 0 in steps of 0.01. An error matrix is created to store the value of error for each value of the pair (A,B). The mean squared error is calculated as follows:

```
for i in range(len(A)):
    for j in range(len(B)):
        for k in range(len(file[:,1])):
            err[i][j] += (1/101) * ((file[k,1] - g1(t[k], A[i], B[j])) ** 2)
        j += 1
    i += 1
```

9 Task 8

A contour plot is plotted showing the error for different values of A and B. It can be seen that there is a minima as all the contours seem to centre around a certain value.

10 Task 9

Here we will be using the built in function `scipy.linalg.lstsq()` to find the values of A and B corresponding to the least mean square error. Firstly, we had computed the matrix M in the function `g1(t, A, B)`. We use the same agorithm here to compute the M matrix. The function `scipy.linalg.lstsq(M,p)[0]` computes the best approximation of the vector x satifying the equation

$$Mx = p$$

When we pass M and the first column of the file as arguments to the function, we get the best estimate of A and B

11 Task 10

Here, we will compute and plot the error between the predicted value and the actual value of A and B.

11.1 Populating the matrices

Four vectors **A, B, errA, errB** are declared with dimensions (columns-1) where columns = no. of columns in the file. Then, the arrays are populated using the following code:

```
[A,B] = scipy.linalg.lstsq(M, file[:, range(1,columns)])[0]
err_A = (A - 1.05) ** 2
err_B = (B + 0.105) ** 2
```

11.2 Plotting the error

Here, in order to get the scattered plots, we use the method `plt.scatter()`. First the graph between the standard deviation and error of A is plotted, followed by the plotting of standard deviation and error of B. Labels for x and y axes and legend are all given appropriately.

12 Task 11

The same plot done in Task 10 is done in a log-log scale here. Errorbars are drawn as well, as shown in the following code:

```
errorbar(scl, err_A, std(err_A), fmt='ro')
errorbar(scl, err_B, std(err_B), fmt='bo'
```

13 Inference

This assignment has been a good learning experience for me as I get to explore a new class of function called Bessel's function. Adding noise to the data and analysing the noise is something which I have never done before in a rigorous sense. Different forms of graphing like contour, scatter plots, loglog scaling have also been the learnings for me via this assignment. Overall, I got to know and explored different features of Python in this assignment.