Assignment 3: Fitting Data to Models

Ayush Jamdar EE20B018

February 18, 2022

1 Aim

The goal of this assignment, briefly described, is to:

1. Read data from a file and parse it.
2. Analyse the data to extract information.
3. Study the effect of noise on the fitting process.
4. Plot graphs in Python.

2 The Assignment

2.1 Introduction

The Python script `generate_data.py` generates a set of data which is written out to a file named `fitting.dat`. I load the file and extract the 10 columns of this data for further operations.

```
time = np.loadtxt(fname='fitting.dat', usecols=0)
data = np.loadtxt(fname='fitting.dat', usecols=(range(1, 10)))
```

2.2 Question 2: Extracting data

The data is a matrix with 10 columns and 101 rows. The `usecols` parameter of `loadtxt` gives the required columns. The first column is time stored in the Python variable `time`. Other nine columns are stored in `data`. The 101 rows correspond to times 0.0 to 10.0. What the data columns are, will be clear in the following sections.

2.3 Question 3: Noise

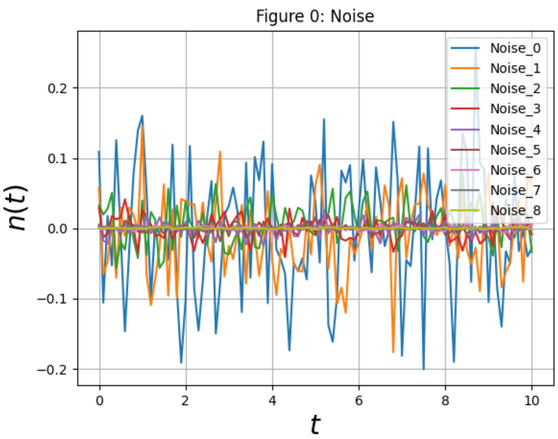
The data columns correspond to the function

$$f(t) = 1.05J_2(t) - 0.105t + n(t)$$

with different amounts of noise added. The noise is given to be normally distributed, i.e., its probability distribution is given by

$$Pr(n(t)|\sigma) = \frac{\exp(\frac{-n(t)^2}{2\sigma^2})}{\sigma\sqrt{2\pi}}$$

with σ given by `sigma=logspace(-1, -3, 9)`
The noise signals $n(t)$ are plotted in Figure 0. The nine $f(t)$ functions for different noise signals are plotted. Please note that my `data` is a 2D numpy array of shape 101x9. So each column represents the `True Value` combined with a different noise distribution. Accordingly, the matrix transpose `data.T` is used in the Python code wherever necessary.



2.4 Question 4: Fitting a function to data

The following function is to be fitted to the data obtained:

$$g(t, A, B) = AJ_2(t) + Bt$$

I have created the following Python function `g(t, A, B)` that computes $g(t, A, B)$ for given A and B. It returns a numpy array of dimensions (101, 1) in our case.

```
def g(t, a, b):
 return np.array(a * sp.jn(2, t) + (b * t))

g1 = g(time, 1.05, -0.105)
```

The plot in Figure 1 shows this function `g1` fitted with all noise distributions (and the `True Value` without noise) for $A = 1.05$ and $B = -0.105$.

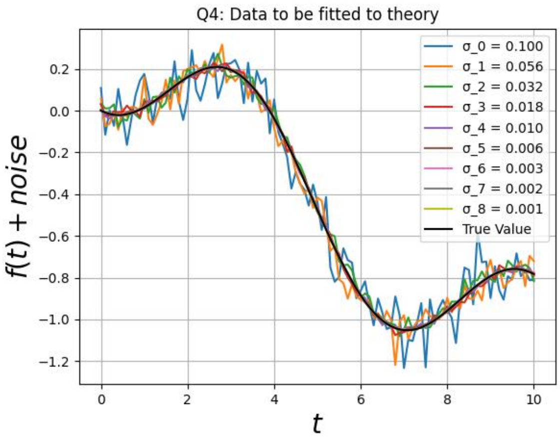


Figure 1: Question 4

2.5 Question 5: Error analysis

Here, I have tried to analyse how much the data diverges due to noise. For the same, I have used the first column of `data`, and plotted how much every fifth data value diverges from the `True Value` by using an `errorbar` graph. The following lines do this.

```
plot(time, g1, 'black', label="f(t)")
errorbar(time[::5], data.T[0][::5], sigma[0],
 fmt='ro', label='Error    bar')
```

The plot can be observed in Figure 2.

2.6 Question 6: Matrix method to find g

The same function $g(t, A, B)$ obtained above for $A = 1.05$ and $B = -0.105$ can also be obtained using matrix product. I have built a matrix M of dimensions (101, 2), the first column of which is the value of the Bessel function for the times in column two. The below lines of code do this.

```
x = np.array([sp.jn(2, time[i]) for i in range(len(time))]).T
y = time.T
M = c_[x, y]
A0 = 1.05
B0 = -0.105
```

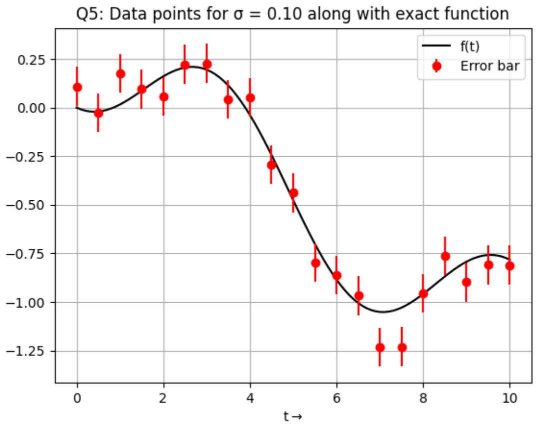


Figure 2: Question 5

```
g2 = dot(M, np.array([A0, B0]).T)

if g2.all() == g1.all():
 print("Part 6: Vectors are equal")
else:
 print("Part 6: Vectors are unequal")
```

I have compared it with the previously defined function (vector) `g1` using a special Python method `.all()`. As expected the code prints:
`Part 6:  Vectors are equal`

2.7 Question 7: Mean Squared Error

For $A = 0, 0.1, \ldots, 2$ and $B = -0.2, -0.19, \ldots, 0$ Mean Squared Error is calculated as

$$\epsilon_{ij} = \frac{1}{101}\sum_{k=0}^{101}(f_k - g(t_k, A_i, B_j))^2$$

In Python,

```
A = arange(0, 2.1, 0.1)
B = arange(-0.2, 0.01, 0.01)
mean_squared_error = []
for i in range(len(A)):
  mean_squared_error.append(zeros(len(B)))

mean_squared_error = np.array(mean_squared_error)
for i in range(len(A)):
```

```
    for j in range(len(B)):
        for k in range(101):
            mean_squared_error[i][j] += (data.T[0][k] -
            g(time[k], A[i], B[j])) ** 2

mean_squared_error /= 101
```

Finally, ϵ_{ij} will be a numpy array of shape (`len(A)`,`len(B)`), where each matrix element is the mean squared error for the corresponding A_i and B_j.

2.8 Question 8: The Contour Plot

As discussed in the previous section, ϵ_{ij} is a matrix with elements based on A and B. So to understand the variation of mean squared error with A_i and B_j, one would need a 3D plot. But thinking of it, a contour plot is also a good way to analyse because of its readability in a 2D plane. Hence, we plot the contour (See Figure 3) with the Python code:

```
cs = contour(A, B, mean_squared_error)
clabel(cs, fontsize=10)
p = scatter(A0, B0) # to plot the exact point
annotate("({}, {})".format(A0, B0), (A0, B0))
```

$A = 1.05$ and $B = -0.105$ are the exact values and have been marked in the plot. ϵ_{ij} has a minimum as can be understood from the plot's contour lines.

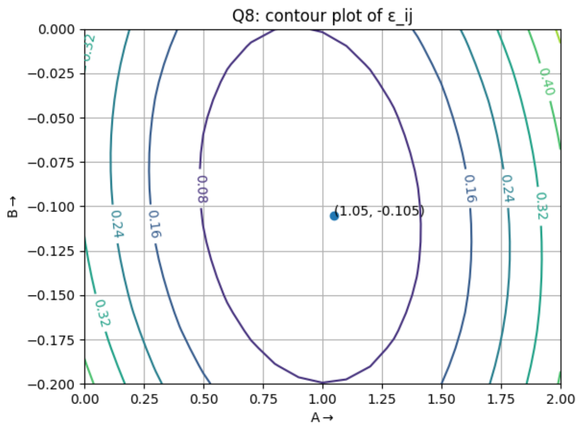


Figure 3: Question 8

2.9 Question 9 and 10: Best Estimate

The lstsq function basically solves a matrix equation $Ax = b$ by computing a vector x that minimizes $|b - Ax|^2$. The equation may be under-, well-, or over-determined (i.e., the number of linearly independent rows of a can be less than, equal to, or greater than its number of linearly independent columns). If a is square and full rank, then x (but for round-off error) is the "exact" solution of the equation. If not, then x gets minimized to Euclidean 2-norm. Here, x is the column (A_i, B_j). The code required is:

```
best_estimate = np.array([np.linalg.lstsq(M,
 (data.T[i]).T)[0] for i in range(9)])
error_in_A = A0 - best_estimate[:, 0]
error_in_B = B0 - best_estimate[:, 1]
```

In each iteration of the list comprehension, I will get the minimized norm of A and B (The column vector of part 6) for each data column. Thus best_estimate will have nine list elements $[A, B]$. Hence the columns of best_estimate when subtracted from A0 and B0 will give the error (with sign; I only need magnitude for the plots) arrays. The same is plotted in Figure 4, that demonstrates variation of these errors with varying noise. The plot is quite linear for $\sigma > 0.02$, but not below it.

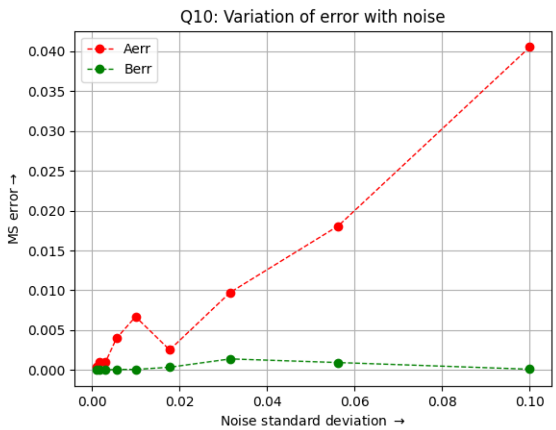


Figure 4: Question 10

2.10 Question 11: The `loglog` plot

The same curves of Error vs Noise Standard Deviation are here plotted on a loglog scale. See Figure 5. These can also be considered to be linear to an extent.

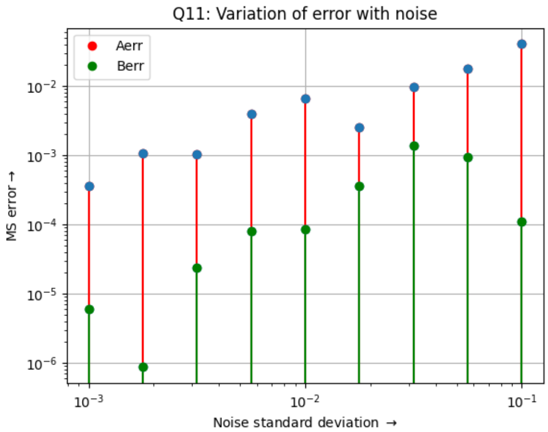


Figure 5: Question 11

3 Conclusion

The aims that I enlisted at the beginning of this report are now seen to have been accomplished. Through every subsection, I analysed how noise and its distribution affects the main signal (function). In the later parts I studied how error is affected by σ. Through this analysis I learnt how `matplotlib` plotting functions work and how data fitting is really done. Some other inferences specific to certain subsections have been explained up there.