EE2703:
APPLIED PROGRAMMING LAB

Assignment 3

Fitting Data to Models

Arjun R
EE20B016

February 18, 2022

Aim

This assignment mainly focuses on

- Reading data from files and parsing them
- Analysing the data to extract information
- Studying the effect of noise on the fitting process
- Plotting graphs for better data visualization

Introduction

At the start of the assignment, a linear combination of the **Bessel function** (of the first kind of real order and complex argument) and the argument t is created as vector $f(t)$ and noises of varying standard deviations are added to it to simulate real-life data.

All of this is done by `generate_data.py` file and the data is stored inside `fitting.dat` file, with the shape of the data being 101 rows X 10 columns. The first column of the `fitting.dat` file is time, and the rest 9 columns contain the function:

$$f(t) = 1.05J_2t - 0.105t + n(t)$$

where J_2t is the $bessel function$ and $n(t)$ is the normally distributed noise, the probability distribution of which is given by:

$$Pr(n(t)|\sigma) = \frac{1}{\sigma\sqrt{2\pi}}exp(-\frac{n(t)^2}{2\sigma^2})$$

Here, σ is generated by function *logspace* from *pylab* module as follows:

```
sigma = logspace(-1, -3, 9)
```

This returns an numpy array of 9 evenly spaced samples in the logarithmic scale from 10^{-1} to 10^{-3}, both inclusive.

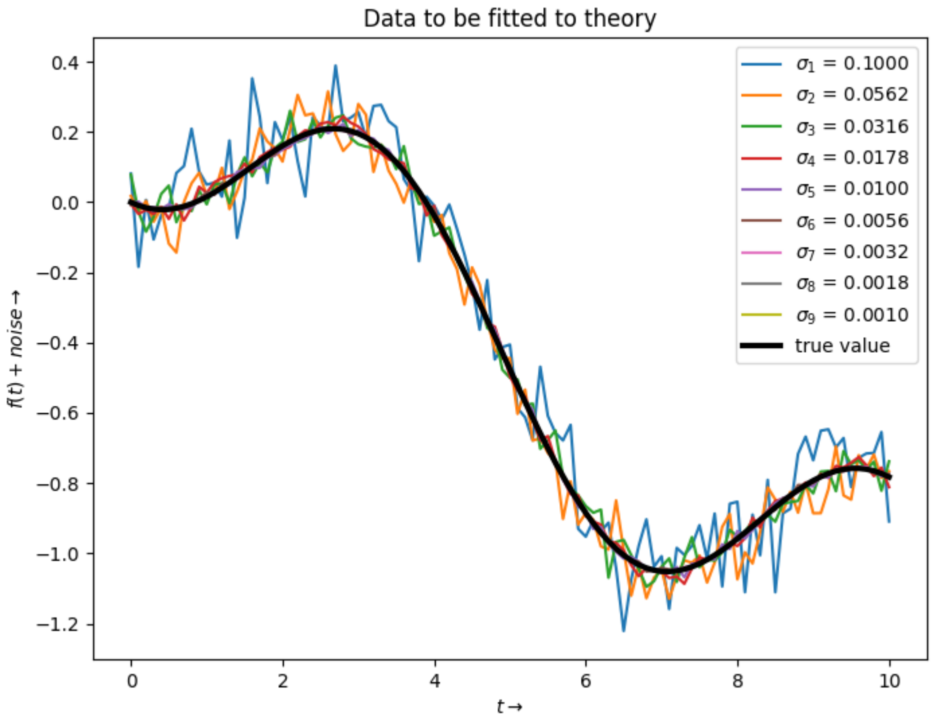

Figure 1: Plot of all function data with varying σ

Error Bar Plot

Error bar plots are useful to visualize the spread of the data around the mean. Here, in figure 2, we plot the error bars to $\pm\sigma$ about the data points and also plot the true function value to see how much the data diverges.

Linear Fitting to Data

The next part of the assignment focuses on modelling of the acquired data using a function that has the same shape as the data but with unknown coefficients:

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

This can be done using the following Python block:

```
import scipy.special as sp
def g(t, A, B):
    y = A*sp.jn(2,t) + B*t
    return y
```

Here, *jn()* function calculates the Bessel function.

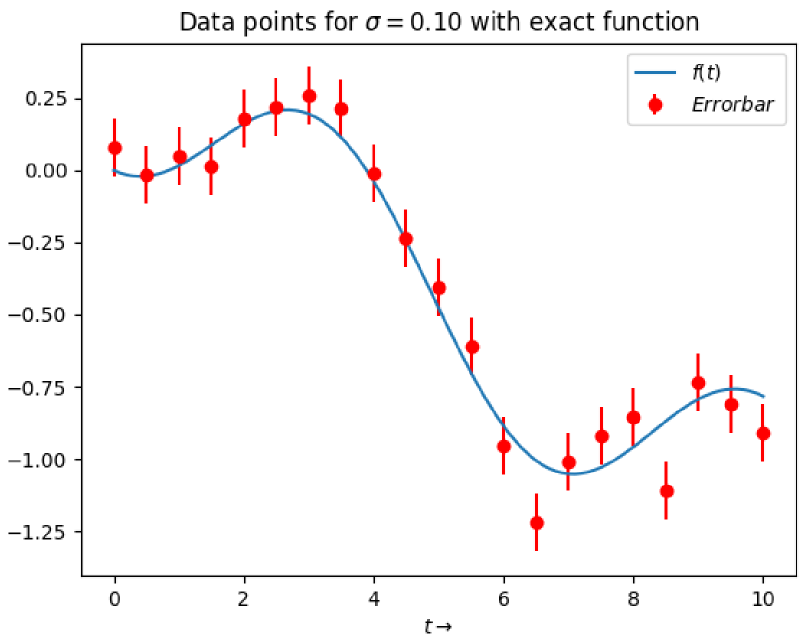

Figure 2: Error bar plot

Since our model is linear in the parameters and the values of t are discrete and known, our task of finding the unknown coefficients can be reduced to inversion of a matrix problem.

Creating the matrix

We can obtain $g(t, A, B)$ as a column vector by creating a matrix equation as follows:

$$g(t; A, B) = \begin{pmatrix} J_2(t_1) & t_1 \\ \dots & \dots \\ J_2(t_m) & t_m \end{pmatrix} \cdot \begin{pmatrix} A \\ B \end{pmatrix} \equiv M.p$$

Mean-squared Error Calculation and Contour Plot

Mean-squared error (MSE) is calculated between the data f_k and the assumed model $g(t; A, B)$ for $A = 0, 0.1, \ldots, 2$ and $B = -0.2, -0.19, \ldots, 0$. Mathematically,

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

The corresponding Python code implementation:

```
import numpy as np
A_arr = np.arange(0, 2.1, 0.1)
B_arr = np.arange(-0.2, 0.01, 0.01)
MSE = np.zeros((len(A_arr), len(B_arr)))
for i in range(len(A_arr)):
    for j in range(len(B_arr)):
        MSE[i, j] = (1/N)*sum(np.square(data[:, 1][k]
                - g(t[k], A_arr[i], B_arr[j]))
                for k in range(N))
```

As the next step, we plot the contour plot for this MSE with A and B as co-ordinates. (We will take the second data column(with $\sigma = 10^{-1}$) as f_k for this purpose)

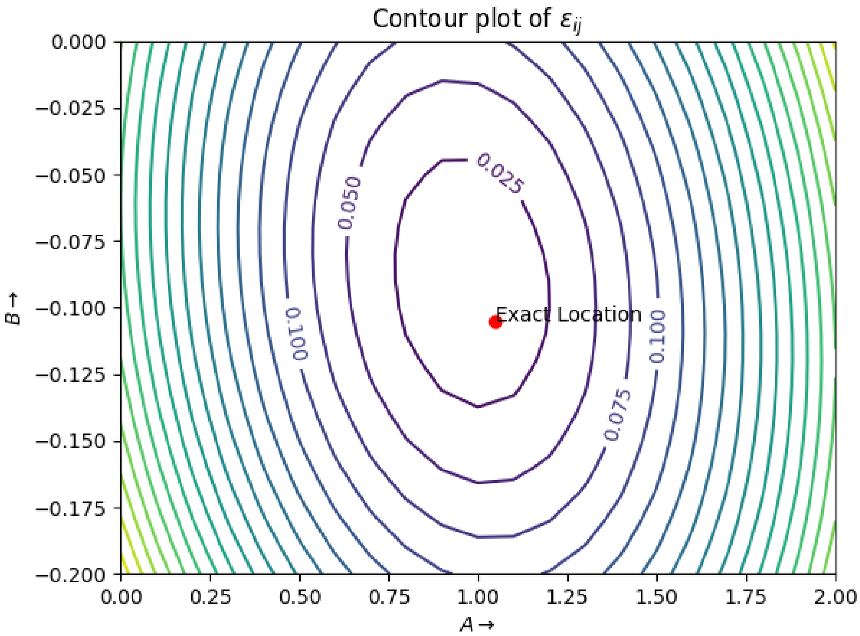

Figure 3: Contour plot for MSE

Figure 3 gives us a clear picture of how the minimum of MSE error closely coincides with the exact location of the true A and B values.

This method can be directly used by calling *lstsq* function from *scipy.linalg* module as follows:

```
from scipy.linalg import lstsq
error = np.zeros((9, 2))
for col in range(1, 10):
    p, resid, rank, sig = lstsq(np.c_[sp.jn(2, t), t],
    data[:, col], rcond = None)
for p_i in range(2):
    error[col-1, p_i] = abs(p[p_i]-p_vec[p_i])
```

Here, the *error* variable will store a $9X2$ numpy matrix with each row storing absolute differences between best estimates of A and B parameters and their true values respectively, for each of the nine function data columns.

These error values are plotted w.r.t. standard deviations of generated noise.

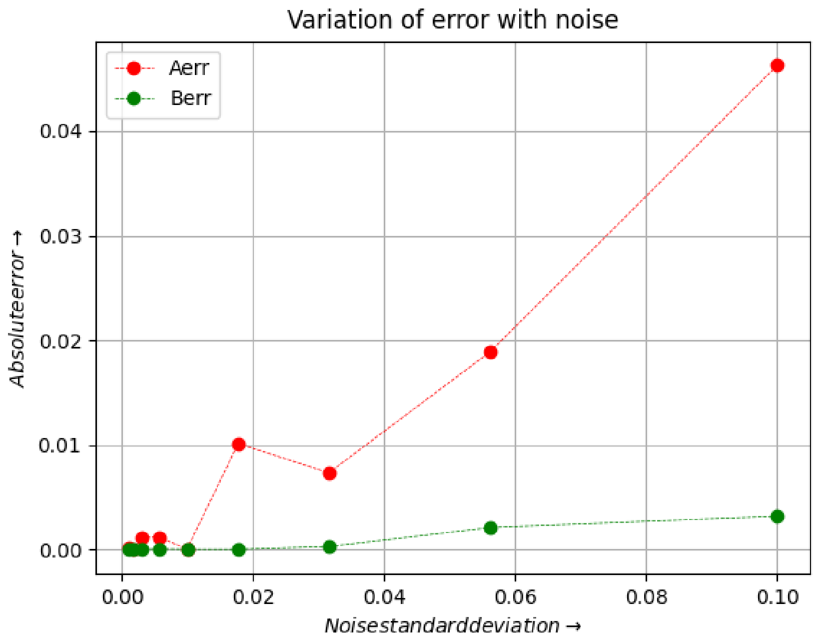

Figure 4: Variation of error in A and B w.r.t. noise

We can clearly see from figure 4 that the increase in error with noise standard deviation is not linear, and this is fairly intuitive since we generated σ in a **logarithmic scale**. Instinctively, we feel that the log-log plot of the same error vs σ plot should give a linear plot.

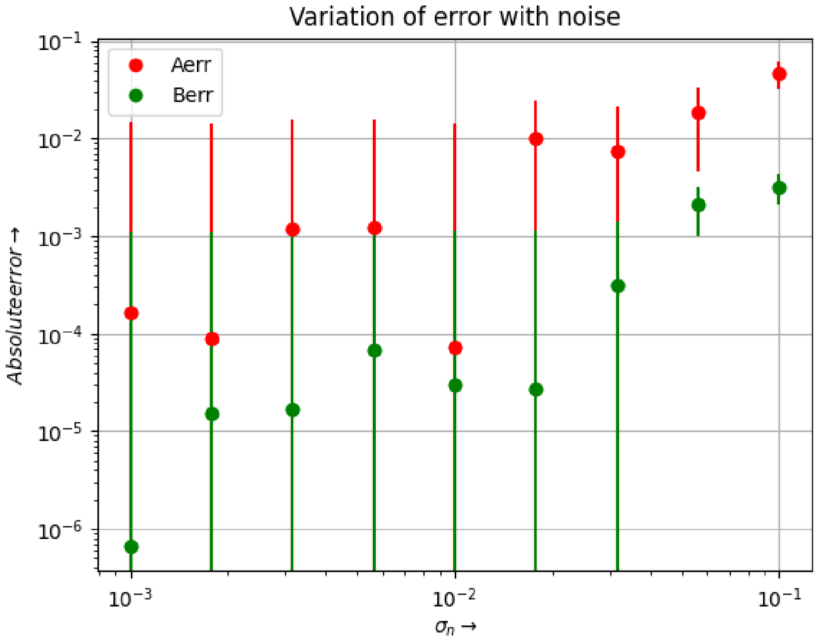

Figure 5: Log-log plot of error variation with noise

From figure 5, we observe that the log-log plot is indeed, *approximately linear*.

Conclusion

We have seen how the best fit can be obtained for a linear model using least-squares method. We have also seen how error values in best fit parameters generated against logarithmically scaled standard deviations in noise, gives an approximately linear log-log plot.