EE2703: Applied Programming Lab
Assignment 3
Fitting Data to Models

Devaganthan S S
EE19B018

March 5, 2021

0.1 Abstract

Aim of this experiment is to,

- Analysing the data to extract information
- Study the effect of noise on the fitting process
- Learn to plot Graphs in python

0.2 Introduction

With the help of computers, it has become very easy to model data. This helps us in predicting things. In this experiment, we try modelling the data, generated using the 'generate_data.py' . With the help of python, we try fitting the data into a function. The function generated is verified by several means.

0.3 Implementation

The data file has to be passed as an argument in the command line, while running the code. The program prints a set of instructions. Based on the requirement, input has to be given

0.4 Results

0.4.1 Generating and Loading Data

Using the 'generate_data.py' , a file named 'fitting.dat' is created, which contains data required for the experiment. The file comprises of 10 columns with 101 rows each. The first column is time. The rest 9 columns are data, with noise, having different standard deviation in the range 10^{-3} to 10^{-1}
The data is loaded into the program using the function 'loadtxt()'

0.4.2 Plotting the Data and True Value

The columns in the data are plotted against time. Each column corresponds to $f(t) + noise$. The noise is not same for all columns. The below code plots the graph in Python.

```
    for i in range(1, 10):  # Plotting the Data against time
         plot(time, data[i], label=scl[i-1])  # Plotted using
    matplotlib.pyplot
     # Plotting the Real Value, i.e without any noise
     plot(time, realV, 'black', label='True Value')
     xlabel('Time')  # X axis Label
     ylabel('f(t)+Noise')  # Y axis Label
     title('Plotting the Data')  # Z axis Label
     legend()
     show()
```

The Output Graph is below,

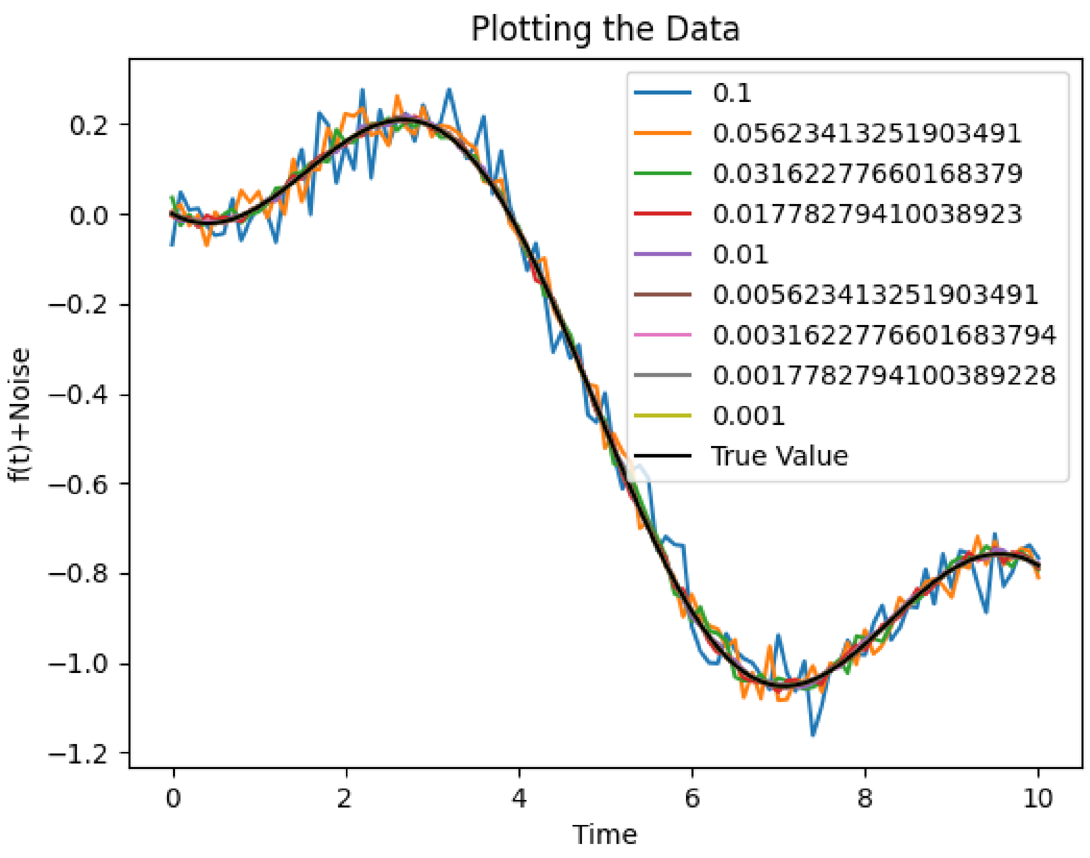

Figure 1:

0.4.3 Plotting the Error Bars:

The error bars are plotted using the function 'errorbar()' , for every 5th item of the first column. Also, the exact curve is plotted to understand how much the data diverges. The below code accomplishes the above,

```
plot(time, data[1], label=scl[0])  # Plot for first Column
plot(time, realV, label='True Value')  # PLotting the Exact Curve
# PLacing the error bars in the graph
errorbar(time[::5], data[1][::5], scl[0], fmt='ro', label='Error
    Bar')
xlabel('Time')  # X axis Label
ylabel('f(t)+Noise')  # Y axis Label
title('With ErrorBar')
legend()
show()
```

The Output Graph is below,

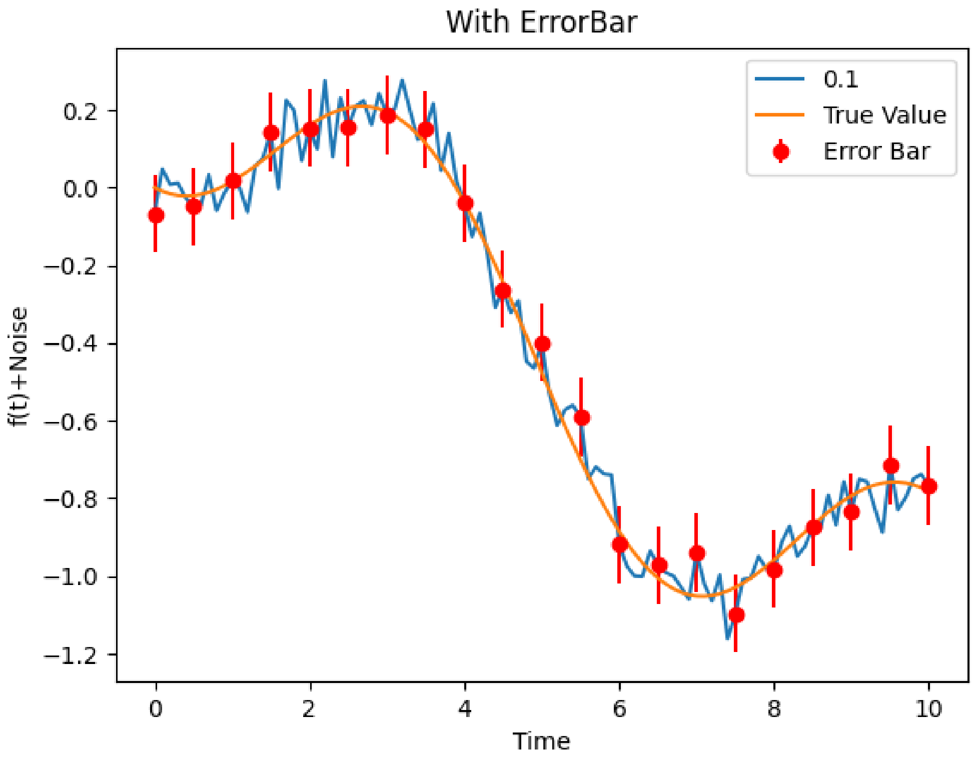

Figure 2:

0.4.4 Constructing the Coefficient Matrix:

The M matrix is generated, with $J(t)$ as its first column and time as its second column. The Vector $M.\begin{pmatrix} A_O \\ B_O \end{pmatrix}$ is generated using the 'dot()' function, where A_O is 1.05 and B_O is -0.105. With the $g(t, A, B)$ function, a column vector is generated with A and B as 1.05 and -0.105 where t is the time vector. The two columns generated are equal and they are verified by taking a sum of squared differences, and it turns out to be zero. The below code accomplishes the above tasks.

```
for i in time:
    realV = realV + [g(i, 1.05, -0.105)]
data = vstack((data, realV))
JCol = []
for i in time:
    JCol = JCol + [sp.jn(2, i)]
M = c_[JCol, time]  # M matrix is created
b = array([1.05, -0.105])  # A0 and B0
print('M = ', end='')
print(M)
print('b = ', end='')
print(b)
print("Sum of squared differences = ", end='')
# Computing the sum of square differences
Error = ((dot(M, b) - realV)**2).sum()
print(((dot(M, b) - realV)**2).sum())
if Error == 0:  # Verification
    print('Verified they are Equal')
```

0.4.5 Mean Square Error Matrix and Contour Plot

For $A = 0, 0.1, \ldots\ldots, 2$ and $B = -0.2, -0.19, \ldots\ldots, 0$ for the data given in Column 1, the mean squared error is computed which is given by,

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

The Mean Squared Error is stored as a Matrix given by 'E'. With this matrix, a contour plot is generated to analyze errors produced by different values of A and B. The above is accomplished by the below code

```
A = arange(0, 2, 0.1)
B = arange(-0.2, 0, 0.01)
```

```
f = data[1]
gcol = []
k = 0
E = []
for i in A:
    temp1 = []
    for j in B:
        gcol = []
        for k in time:
            gcol = gcol + [g(k, i, j)]
        temp = subtract(f, gcol)
        temp = multiply(temp, temp)
        temp = sum(temp)/101
        temp1 = temp1 + [temp]
    E = E + [temp1]
clabel(contour(A, B, E, levels=20))
xlabel('A')
ylabel('B')
title('Contour of Error ij')
show()
```

The Output Graph is below,

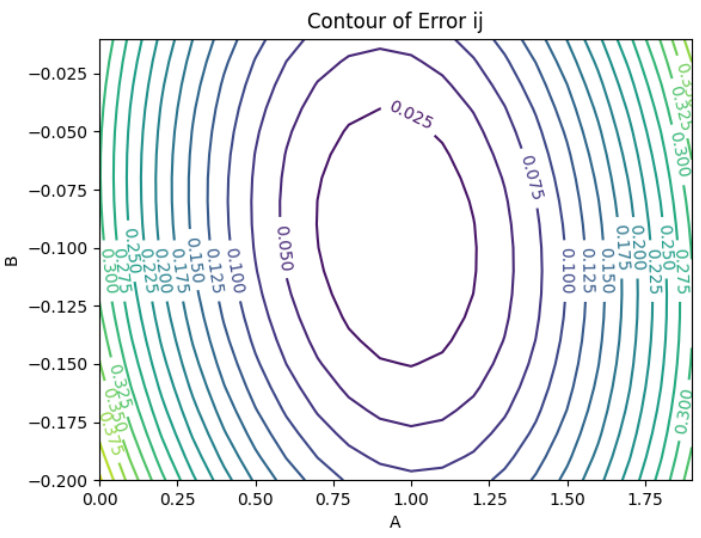

Figure 3:

From the Contour plot, we can see that, there is only 1 minimum. The graph seems to have a minimum near A = 1.05 and B = -0.105.

0.4.6 Best Approximate for A and B

The best approximate for A and B is computed for each data Column using the 'lstsq()' function from 'scipy.linalg' . The above is accomplished by the below code.

```
JCol = []
for i in time:
    JCol = JCol + [sp.jn(2, i)]
M = c_[JCol, time]
solution = []
for i in data:
    solution = solution + [lg.lstsq(M, i)[0]]
solution = array(solution)
print('Best Estimate of A and B =')
print(solution)
```

0.4.7 Plotting Error in the Estimate

The error in the estimate of A and B is computed using the mean Square method for data files which has noises of different standard deviations. Then the estimate of A and B are plotted as a function of the standard deviation, σ. The below code accomplishes the above,

```
error = []
for i in solution[:-1]:
    error = error + [array([(i[0]-1.05)**2, (i[1]+0.105)**2])]
error = array(error)
if '10' in reqPlots:
    plot(scl, error[:, 0], 'o', linestyle='dashed', label='A')
    plot(scl, error[:, 1], 'o', linestyle='dashed', label='B')
    xlabel('Noise Standard Deviation')
    ylabel('Error')
    legend()
    title('Variation of Error With Noise')
    show()
```

The Output Graph is below,

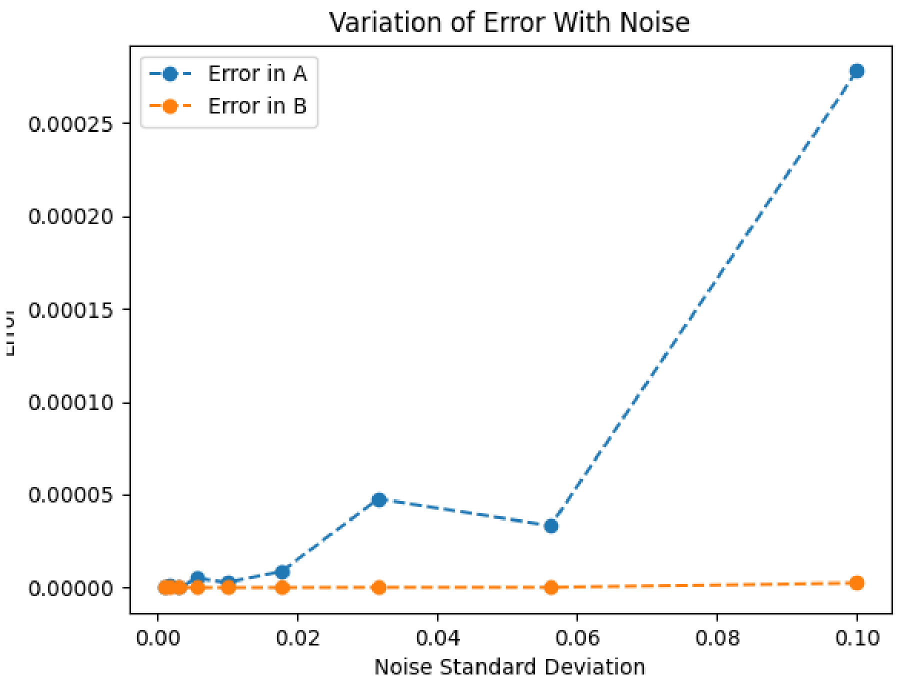

Figure 4:

From the graph, we can notice that the error seems to increase linearly with the standard deviation.

The same plot is redone using 'loglog()' to display the plot in logscale. The below code accomplishes it,

```
error = []
for i in solution[:-1]:
    error = error + [array([(i[0]-1.05)**2, (i[1]+0.105)**2])]
error = array(error)
loglog(scl, error[:, 0], 'o', linestyle='dashed',
       label='Aerr', markersize='4')
loglog(scl, error[:, 1], 'o', linestyle='dashed',
       label='Berr', markersize='4')
xlabel('Noise Standard Deviation')
ylabel('Error')
legend()
title('loglog error vs stdev')
show()
```

The Output Graph is,

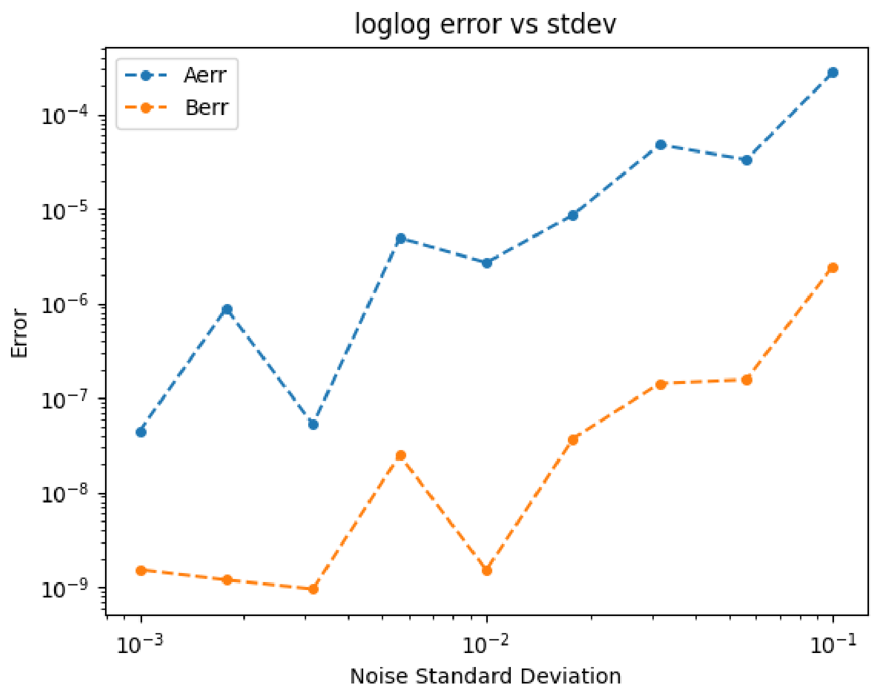

Figure 5:

The error is approximately linear with respect to standard deviation except for a few points.

0.5 Conclusion:

From this experiment, I have learned how to plot functions in python. I learnt how to model a given set and compute the error in the Model.