EE2703: Applied Programming Lab

Assignment 3: Fitting Data To Models

Purvam Jain EE20B101
Electrical Engineering

February 18, 2022

Abstract

This week's assignment is about exploring the least squares method from scipy library and fitting data.The main content of the assignment is:

- Analysing data to extract information out of it.
- To study the effects of noise on the fitting process.
- To plot a number of different types of graphs.

1 Program Structure

- Extraction and Visualization of Data
  Computing and plotting MSE
- Applying *Least squares regression* and analyzing effect of noise

The program can be run from the command line and accepts a input *fitting.dat* file as follows:-

```
python EE20B101.py path/to/fitting.dat
```

2 Extracting and Visualizing the data

On running the python code *generate_data.py*, it creates a file *fitting.dat*. This gives rise to the following plot of a function with added noises.The black line represents the actual function.

This file contains 10 columns with 101 rows of data. The first column is the time values and the next nine rows are the noisy values of a function as shown above. Each column of data has its own standard deviation which is given by the python command:

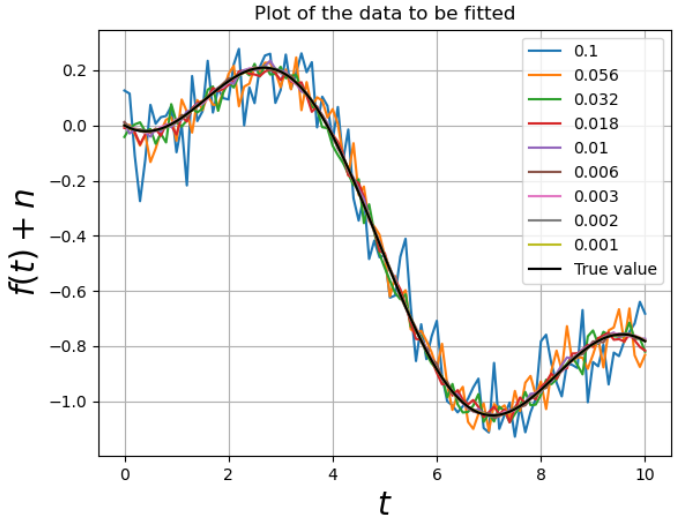

Figure 1: Data Visualization plot

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

Since, the actual function is known, we can plot it's graph also. The function is defined in python as the following code snippet:

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

On plotting the true function's value along with all the 9 noise added values, the plot was generated. This is the Figure 1 that was asked in Q.3 and Q.4. The python code snippet for plotting the follwing graph is as follows:

```
    pylab.plot(t,yy)
    pylab.plot(t,g(t,true_A,true_B),color='black')
    pylab.xlabel(r'$t$',size=20)
    pylab.ylabel(r'$f(t)+noise$',size=20)
    pylab.title(r'Plot of the data to be fitted')
    legends = list(np.around(sigma,3))
    legends.append("True value")
    pylab.legend(legends)
    pylab.grid(True)
    pylab.show()
```

3 Visualising Noise - The Errorbar plot

Error bars help you indicate estimated error or uncertainty to give a general sense of how precise a measurement is this is done through the use of markers drawn over the original graph and its data points. The errorbars for the first data column are plotted using the **errorbar()** function. The python code snippet for plotting the errorbar plot is as follows:

```
pylab.plot(t,g(t,true_A,true_B),color='black')
pylab.errorbar(t[::5],data[:,1][::5],sigma[0],fmt='.')
pylab.title('Error bars for Column 1 data with sigma = 0.1')
pylab.xlabel('t')
pylab.ylabel('f(t) + noise')
pylab.legend(['True Value','error_bar'])
pylab.grid(True)
pylab.show()
```

The graph obtained by plotting every 5th data point with errorbars and the original data is as follows:

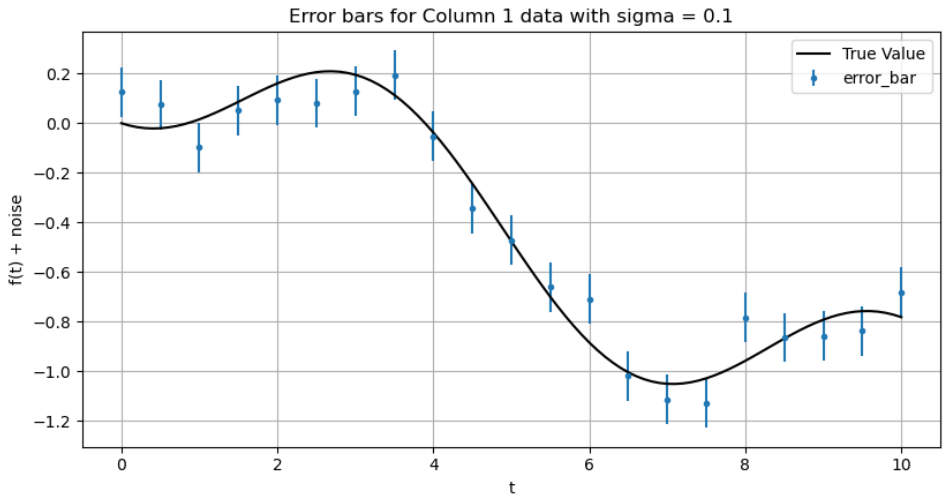

Figure 2: Errorbar plot

4 Matrix Model $g(t, A, B) = M.p$

The actual value function can be created using a matrix equation containing time and $J_2(t)$ values in M and A, B values in p. The matrix M multiplied with (A,B) matrix will result in the actual function. This can be verified by substituting $A = 1.05$ and $B = -0.105$. In order to compare 2 matrices,

we use the function `numpy.allclose`. The python code snippet is as shown below:

```
 = sp.jn(2, t)
M = pylab.c_[J,t]
P0 = np.array([[true_A],[true_B]])
# np.allclose method is used to compare arrays woth floating point dtype to avoid ro
print(np.allclose(np.dot(M,P0),np.reshape(g(t,true_A,true_B),(101,1))))
```

5 The Mean Squared Error

The mean squared error is the error between the noisy data and the true functional data. It is calculated as follows:

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

We used `mean_squared_error` utility from the `sklearn.metrics` library.The python code snippet to calculate the mean squared error is as follows:

```
A = np.linspace(0,2,21)
B = np.linspace(-0.2,0,21)
errors = np.zeros((21,21))
for i in range(len(A)):
    for j in range(len(B)):
        errors[i,j] = mean_squared_error(data[:,1],g(t,A[i],B[j]))
```

The contour plot for ε for various values of A and B is:

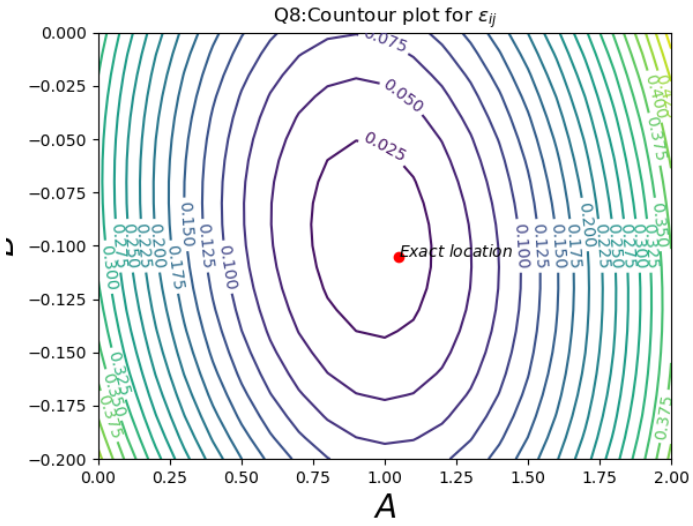

Figure 3: Contour plot

6 Error Analysis

We compute the values for A and B from the matrix M by using the `lstsq()` function form `scipiy.linalg`. Using this we can calculate the error in the values of A and B. Then we can calculate error in the respective values and theirr relation with sigma(standadrd deviation) of the generated noise. The python code snippet is as follows:

```
A_error=[]      # Store error in value of A
B_error=[]      # Store error in value of B
Mean_errors=[]   # To store errors for each data column
for l in range(1,10):
    p,_,_,_ = lstsq(M,data[:,l])

    Mean_errors.append(mean_squared_error(np.dot(M,p),data[:,l]))
    A_error.append(abs(true_A- p[0]))
    B_error.append(abs(true_B - p[1]))
```

The plot of the error in A and B against the noise standard deviation is:

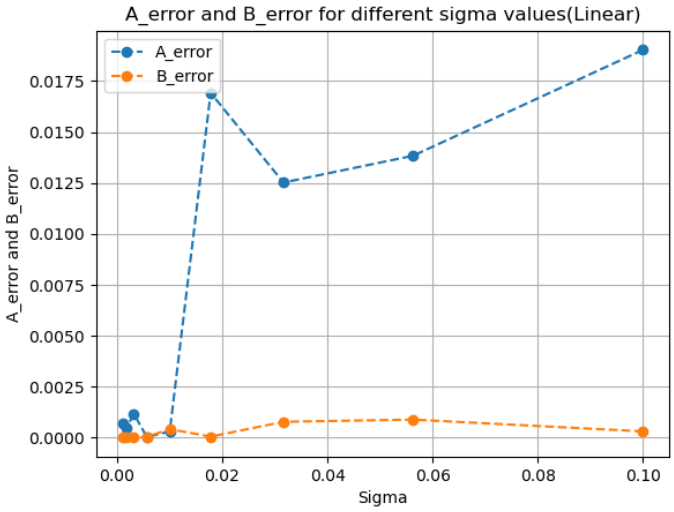

Figure 4: Error vs Standard deviation

We can also plot the same graph in log scale. This plot is shown below:

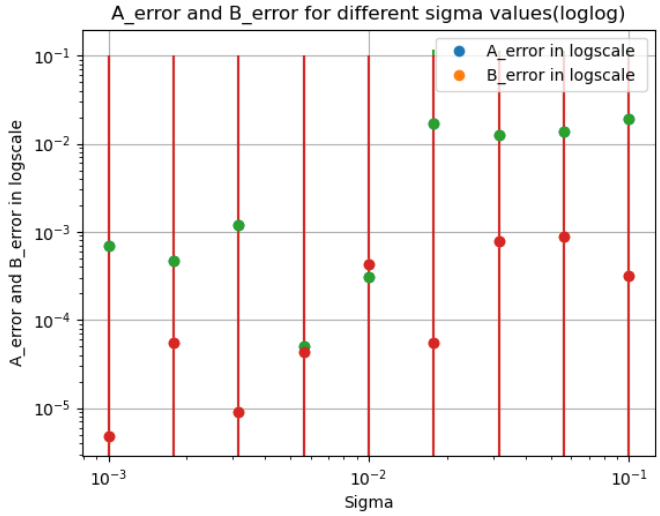

Figure 5: Error vs Standard deviation: log scale

Inference

Below I have plotted the actual `MSError` for every data column with A and B values generated using `lstsq` method against standard deviation for each noisy column in linear and loglog scales. We get almost a linear graph in loglog scale.

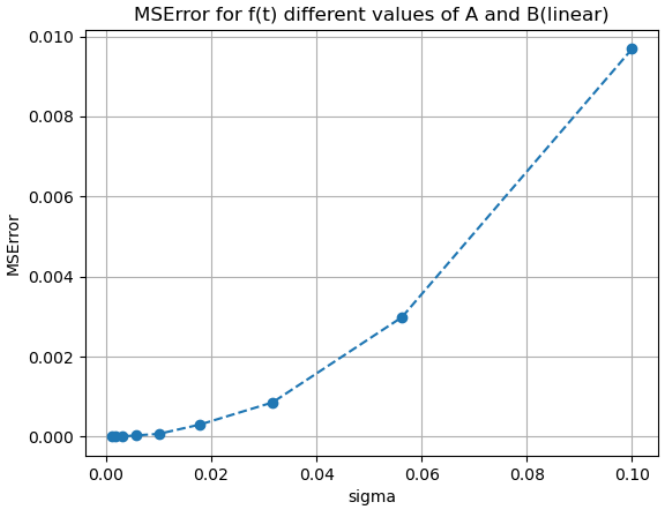

Figure 6: MSError vs Standard deviation: Linear

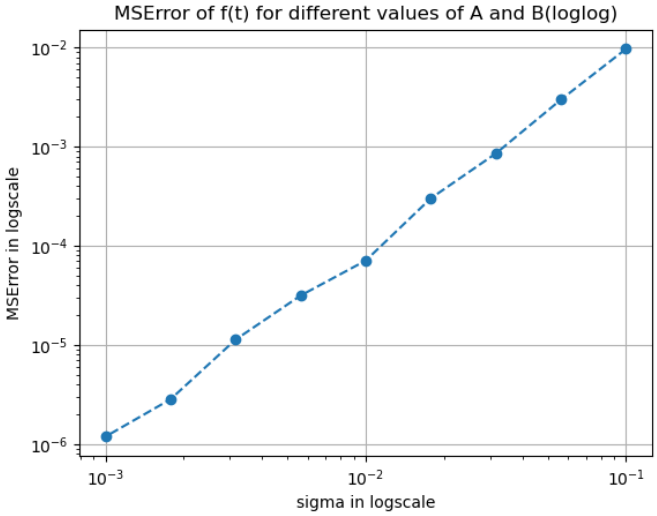

Figure 7: MSError vs Standard deviation: log scale

1. We can see that only one minimum exists in the contour plot.
2. The linear plot between *MSError* and *standard deviation* are porportionate to sigma values are justified since least squares method ignores the zero-centered noises.