Assignment 3: Fitting Data To Models

Aman Kumar EE19B066

March 5, 2021

1 Abstract

This week's assignment is about fitting data to models. The main content of the assignment is:

- Reading data from files and parsing them
- Analysing the data to extract information
- To study the effects of noise on the fitting process.
- To plot a number of different types of plots.

2 Importing the data from file

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.

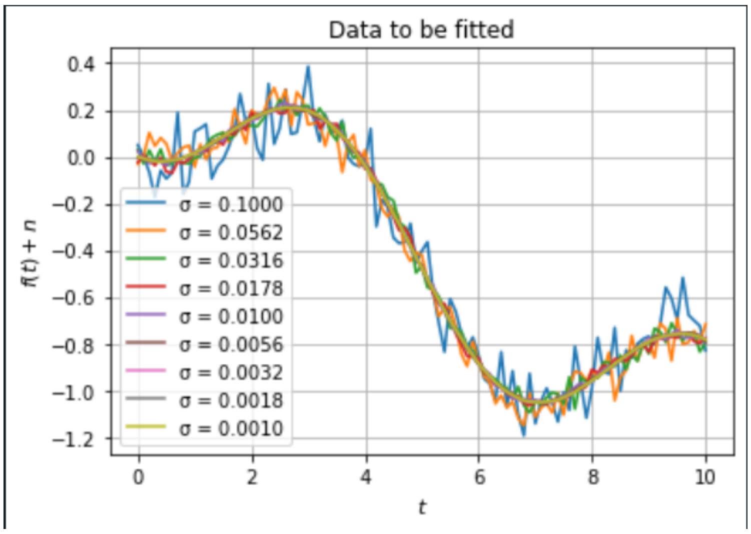

Figure 1: Data to fit to assumed model

This file contains 10 columns with 101 rows of data. The first column is time and the next nine columns are the values of the function with different amount of noise added. Each column of data has its own standard deviation which is given by the python command

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

To extract the data from the *fitting. dat* file python's loadtxt function was used.

```
data = np.loadtxt("fitting.dat")
t = data[:,0]
```

3 The Function with Noisy plots

Model function is given as:

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

Q3. and Q4. asks to plot the nine data columns along with the true value. The plot that is obtained is given below. The part of python code used for this:

```
for i in range(1,10):
    pl.plot(t,data[:,i],label="\u03C3 = %0.4f" % stdev[i-1])
pl.plot(t,y0,label="True Value",color='black',linewidth = 3)
```

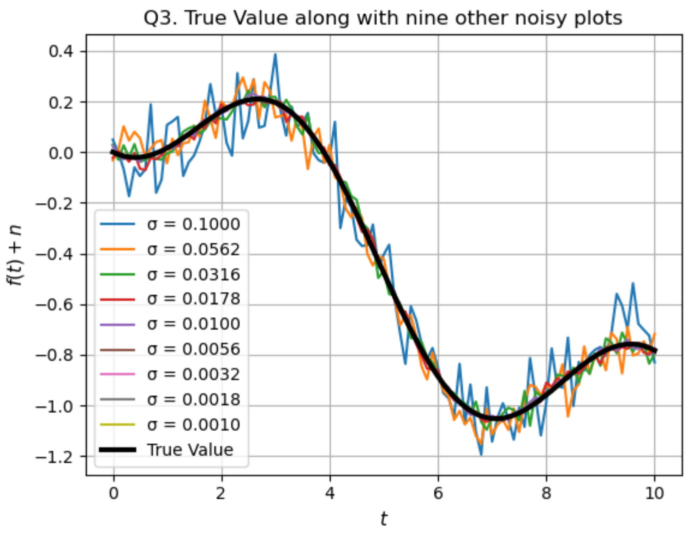

Figure 2: True and noise added plots

4 Plotting the Error Bars

An errorbar is a convenient way of visualising the uncertainty in the reported measurement. The errorbars for the first data column are plotted using the **errorbar()** function. The part of python code used for plotting the errorbar plot::

```
pl.errorbar(t[::5],data[:,1][::5],stdev[1],fmt='ro',label="error bar")
pl.plot(t,y0,label = "True value",color = 'black')
```

The graph obtained by plotting every 5th data point with errorbars and the original data is as follows: See Figure 3 on page 3

5 Generating the matrix

The function can be created using a matrix equation also. The matrix M created when multiplied with (A,B) matrix will be equal to the value of function. This can be verified by substituting $A = 1.05$ and $B = -0.105$. In order to compare 2 matrices, we use the function `array_equal()`. Python code:

```
Jt = sp.jn(2,t)
M = pl.c_[Jt,t]
if (np.array_equal(np.dot(M,A0B0),y0)):
    print("M*[A0;B0] is equal to g(t,A0,B0). Verified!")
else:
    print("M*[A0;B0] is NOT equal to g(t,A0,B0).")
```

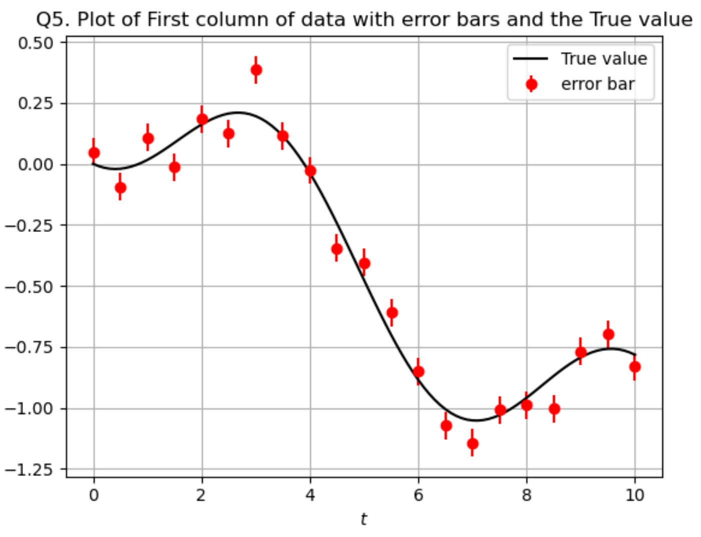

Figure 3: Errorbars with True value

6 Estimating A and B

A and B are estimated by minimising the ε_{ij}. It is done here using **lstsq()** function as given below:

```
def estimate_A_B(M,col):
    AB = pl.lstsq(M,col,rcond=None)
    return(AB[0][0],AB[0][1])
```

7 Mean Squared Error for Different A and B

The mean squared error between the noisy data and the true value is calculated as follows:

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

The python code to calculate the mean squared error is as follows:

```
A = np.linspace(0, 2, 21)
B = np.linspace(-0.2, 0, 21)
Err = np.zeros((21,21))
f1 = data[:,1]
for i in range(21):
    for j in range(21):
        Err[i,j] += np.sum(((f1 - g(t,A[i],B[j]))**2))/101
```

The contour plot for ε_{ij} for various values of A and B is:

7.1 Conclusion

From the above plot, we can conclude that there exist one and only one minimum for $_{ij}$ and as we move away from $A0, B0$ value of ε_{ij} increases

8 Error in the Estimation of A and B

A, B are calculted by minimising ε_{ij} by *lstsq()* function form *scipiy.linalg*. Since, we know $A0, B0$ we can calculate the MSerror in the values of A and B.Python code:

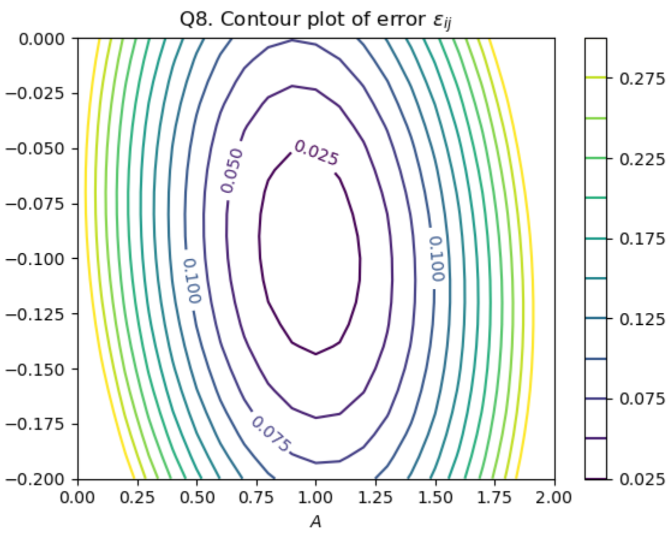

Figure 4: Contour plot of ε_{ij}

```
Aerr,Berr = np.zeros(9),np.zeros(9)
for i in range(1,10):
    A_est,B_est = estimate_A_B(M,data[:,i])
    Aerr[i-1] += ((A_est - A0B0[0])**2)
    Berr[i-1] += (B_est - A0B0[1])**2
```

The plot of the MSerror in A and B vs noise in linear and loglog scales:

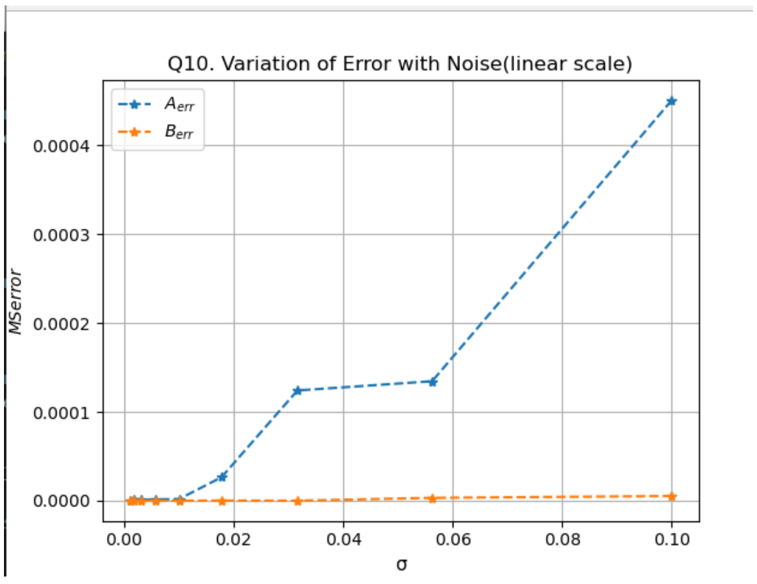

Figure 5: Error vs Standard deviation(linear scale)

The part of code used to plot the graph in Figure 5:

```
pl.plot(stdev,Aerr,marker = "*",label="$A_{err}$",linestyle = 'dashed')
pl.plot(stdev,Berr,marker = "*",label="$B_{err}$",linestyle = 'dashed')
```

and Figure 6:

```
pl.loglog(stdev,Aerr,'ro',label="$A_{err}$")
pl.errorbar(stdev,Aerr,pl.std(Aerr),fmt='ro')
pl.loglog(stdev,Berr,'go',label="$B_{err}$")
pl.errorbar(stdev,Berr,pl.std(Berr),fmt='go')
```

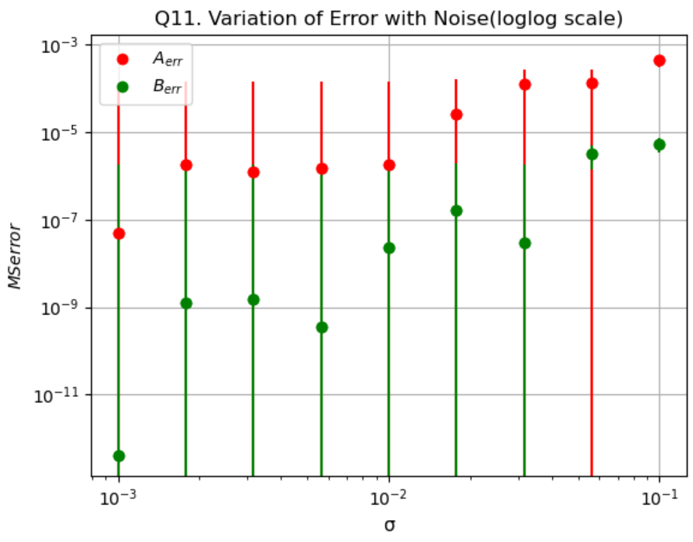

Figure 6: Error vs Standard deviation(loglog)

8.1 Conclusion

From figures 5 and 6 it is evident that the noise is linear in **neither** of the scales. This is because our noise varies on a logarithmic scale and it is expected that the error must also vary in a logarithmic manner.

9 Conclusion

We have used the **lstsq()** function to fit a noisy data to a function. As the noise in the data used to estimate the linear combination increases, the error in the estimation also increases.