# Assignment 3: Fitting Data to Models

J.Phani Jayanth - EE19B026

March 3, 2021

## 1 Abstract

This week's Python Assignment involves the following learning outcomes:

- Obtaining data from files and parsing them
- Analysing the obtained data to extract information
- Effect of noise on the fitting process
- Plotting graphs

## 2 Introduction

We have been given a code to obtain the dataset required. It will be written into a file *fitting.dat*. This data is the noisy version of a given function f(t):

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

where n(t) is the noise added and is normally distributed (Gaussian noise) We are expected to find the best fit of this data and are also required to obtain various plots, as discussed in the following sections.

## 3 Data Analysis and Plots

The data from *fitting.dat* file is extracted and parsed into columns, where the first column corresponds to time and the remaining are noise-added outputs of the given function with different standard deviations($\sigma$). Data in each of the 9 columns is plotted against time, along with the true value of f(t) as reference.

A function 'g' is defined with arguments t,A and B to obtain the true value of the function f(t). *Scipy library*, which includes the Bessel function is used to obtain the true value, which is plotted with the noisy data against time.

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

$$(\text{A} = 1.05,\ \text{B} = \text{-}0.105)$$

The function *g(t;A,B)* is defined in the code as follows:

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

To plot the noisy data and actual function against time:

```
for i in range(9):
plot(n[0],n[i+1], label=f'\u03C3{i+1}={(sigma[i]):.3f}')

plot(n[0],g(n[0],1.05,-0.105),label='True Value',color='k')
```

The plot is obtained as follows:

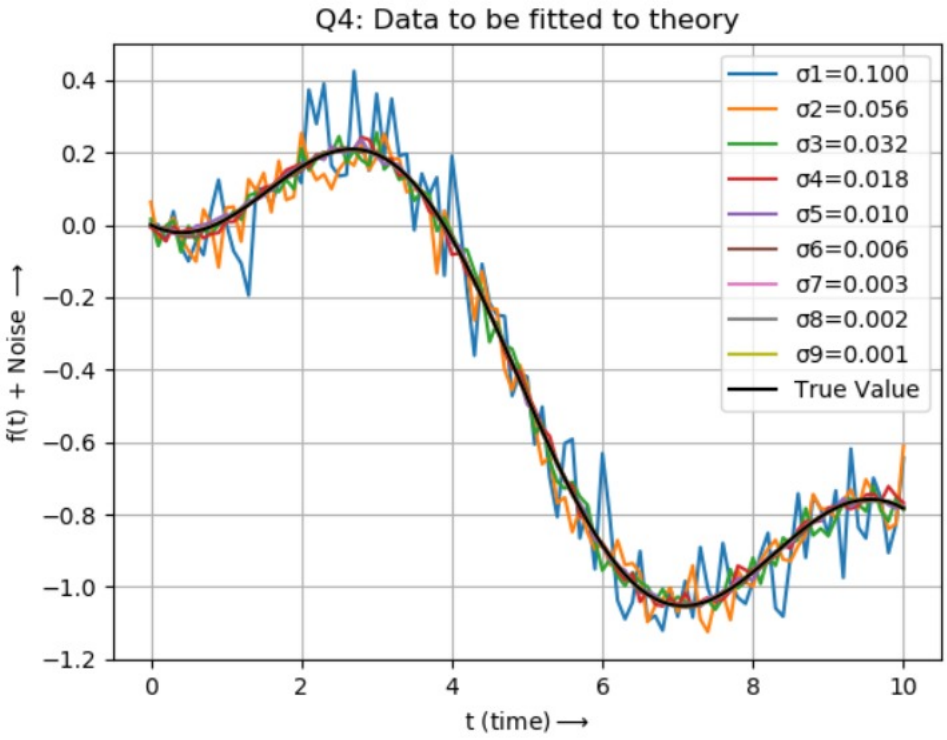


Figure 1: Actual function and Noise-added Data vs Time

### Generating a plot of first column of data with error bars:

Standard deviation of data in the first column is $\sigma = 0.10$. Every fifth data item in this column is taken and plotted with error bar alongside the actual function, from which we can see how much the data diverges from the actual curve. We can plot error bars with red dots using:

```
errorbar(t,data,stdev,fmt=ro)
```

And to show every fifth data point, we use:

```
errorbar(t[::5],data[::5],stdev,fmt=ro)
```

Data points with standard deviation($\sigma$) = 0.10 with error bars, along with the exact function are plotted as follows:

```
errorbar(n[0][::5],n[1][::5],sigma[0],fmt='ro',label='Errorbar')
plot(n[0],g(n[0],1.05,-0.105),label='f(t)',color='k')
```

The plot is obtained as follows:

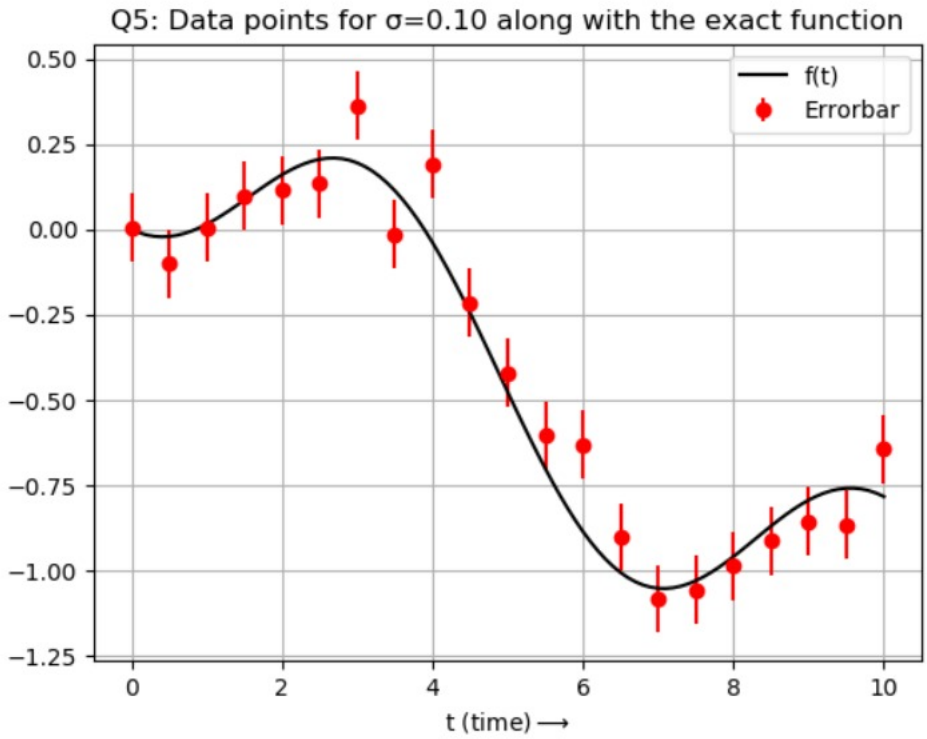


Figure 2: Noisy data with error bars alongside the actual curve vs Time

**Note:** In the code, the data from the file *fitting.dat* is stored in an array 'n', where n[0] corresponds to time and the remaining correspond to data points.

# 4 Matrix equation and Obtaining g(t;A,B) as a column vector

$$g(t;A,B) = \begin{pmatrix} J_2(t_1) & t_1 \\ ... & ... \\ J_2(t_m) & t_m \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} = M.p \tag{1}$$

### Construction of M matrix:

We need to create a matrix M as:

$$M = \begin{pmatrix} J_2(t_1) & t1 \\ \cdots & \cdots \\ J_2(t_m) & t_m \end{pmatrix} \tag{2}$$

When the matrix M is multiplied by

$$p = \begin{pmatrix} A_0 \\ B_0 \end{pmatrix} \tag{3}$$

we should obtain g(t;A,B).

So, M and p are multiplied for $(A_0, B_0)$ = (1.05,-0.105) and a vector is obtained. This vector will be equal to g(t;1.05,-0.105), if the mean of their difference is zero. We achieve this in our code as follows:

```
#Defining matrix M

t = array(n[0])
t=t[:, newaxis]
j2_t = sp.jn(2,n[0])
j2_t= j2_t[:, newaxis]
M=c_[j2_t,t]
p = [[1.05],[-0.105]]

#Verifying whether both these are equal

print('Verifying if M.p = g(t,A,B): at A = 1.05 and B = -0.105\n')
print(f'Error of M.p with actual function: {average(g(n[0],1.05,-0.105)-dot(M,p))}\n
```

When the code is run, this error will be printed and we can see that it is almost zero (Obtained error = -2.229e-17).

### Contour plot of Mean Squared error:

We create A = 0,0.1,....,2 and B = -0.2,-0.19,....,0 as follows:

```
A = arange(0, 2.1, 0.1)
B = arange(-0.2, 0.01, 0.01)
```

We initialize the $\epsilon_{ij}$ as follows:

```
epsilon = zeros((len(A),len(B)),dtype=float)
```

Mean squared error($\epsilon_{ij}$) between the data and the assumed model is calculated using the formula:

$$\epsilon_{ij} = \frac{1}{101}\sum_{k=0}^{101}(f_k - g(t_k; A_i, B_j))^2 \tag{4}$$

This is calculated in the code as:

```
for i in range(len(A)):
for j in range(len(B)):
for k in range(len(n[0])):
epsilon[i][j] += (1/101)*((n[1][k]-g(n[0][k],A[i],B[j]))**2)
```

The contour plot is then plotted as follows:

```
c=contour(A,B,epsilon,20)
clabel(c,c.levels[:5],inline=True)
plot(1.05,-0.105,'ro', label='Exact Location')
title('Q8: Contour plot of \u03B5ij')
xlabel('A $\longrightarrow$')
ylabel('B $\longrightarrow$')
legend()
show()
```

### Finding the values of A and B for minimum Mean squared error:

When the code is run, the minimum error and the values of A and B for which minimum error is obtained will be printed. This is obtained in the code as follows:

```
min_error = amin(epsilon[i][j])
for i in range(len(A)):
for j in range(len(B)):
if (epsilon[i][j]==min_error):
print(f'Minimum error is {epsilon[i][j]} for A = {A[i]} and B = {B[j]}')
```

We obtain a minimum Mean squared error of 0.483 at A = 2.0 and B = 1.665e-16.

**Contour Plot:**

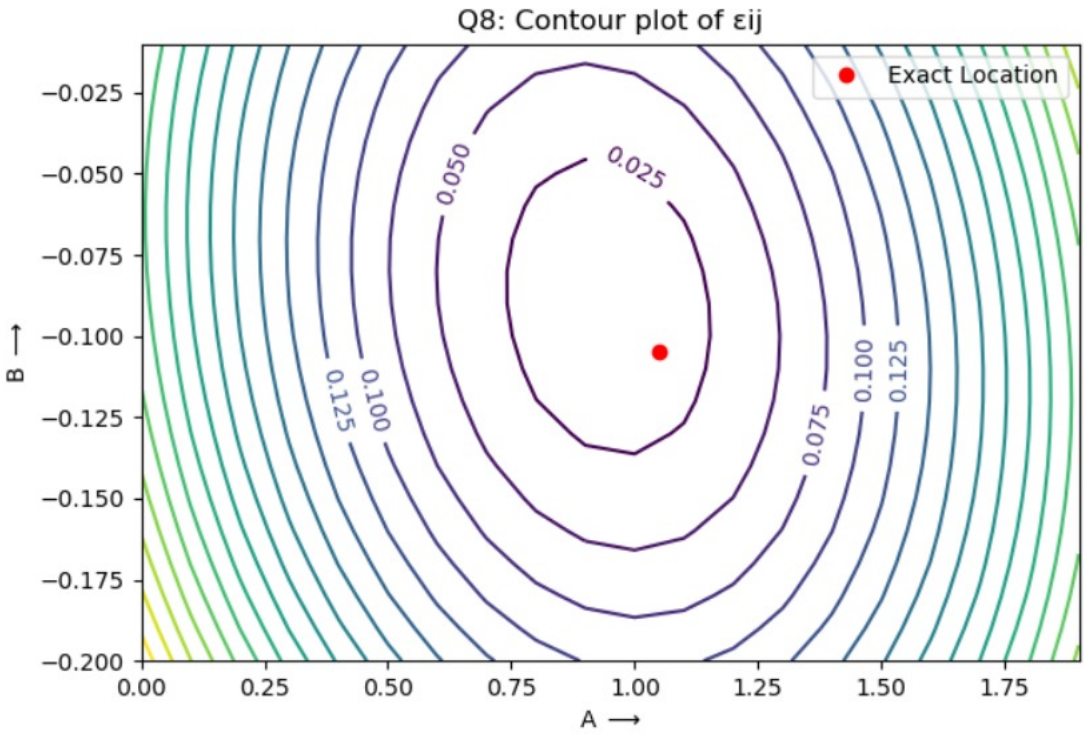


Figure 3: Contour plot representing standard deviation: Red dot representing the True value of function at A=1.05, B=-0.105

## 5 Obtaining best estimate of A and B

lstsq function in *Scipy library* is used to solve for (A,B), when equating M.p with a column of noisy data. Each column, on solving, returns different values of A and B. Error of each of these solutions is plotted against the standard deviation of noise. Plot of Error in B vs Error in A is not a single line. There is no visible trend in the plot of A(error) and B(error).
The code is as follows:

```
h=[]
A_error = []
B_error = []
try: #Solving for best estimate
for i in range(9):
h.append(scipy.linalg.lstsq(M,n[i+1])[0])
```

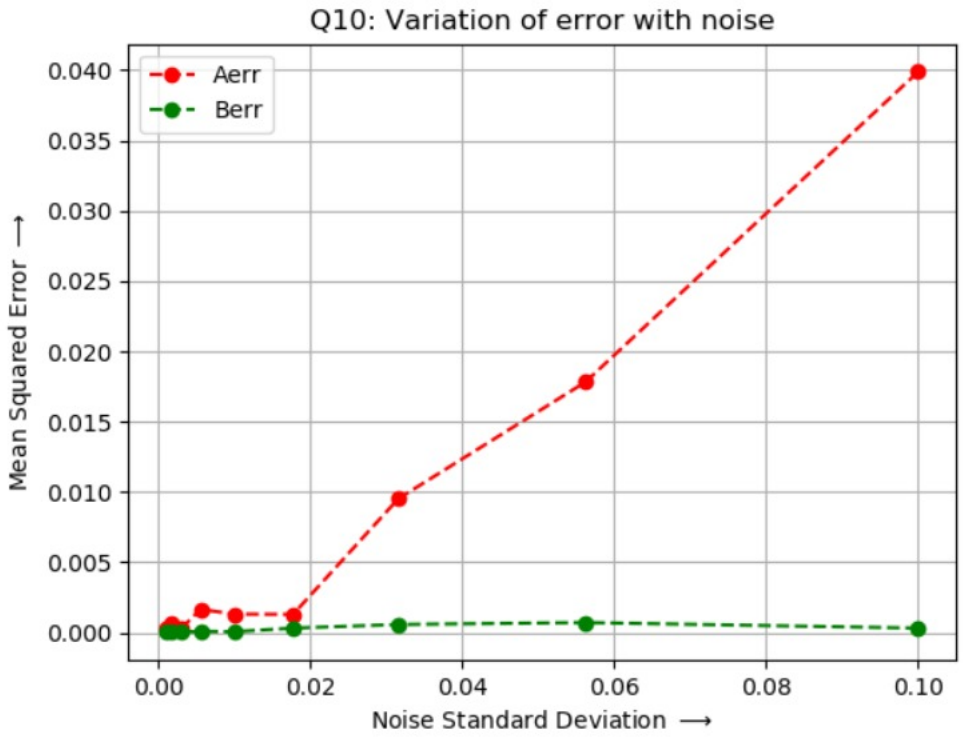


Figure 4: Errors in A and B vs Standard Deviation of noise

```
except Exception:
print("The equation M.p = data couldn't be sovled!")
exit()

print("Solutions for different sigma:")
print(h)
for i in range(9): #Calculating error in A and B
A_error.append(abs((h[i][0]-1.05)))
B_error.append(abs((h[i][1]+0.105)))
#Plotting Aerr and Berr againt noise standard deviation
plot(sigma,A_error,'--r',label = 'Aerr',marker="o")
plot(sigma,B_error,'--g',label = 'Berr',marker="o")
title('Q10: Variation of error with noise')
xlabel('Noise Standard Deviation $\longrightarrow$')
ylabel('Mean Squared Error $\longrightarrow$')
grid()
legend()
show()
```

Error in the in solution of (A,B) is plotted on log-scaled graph, and it is observed that it almost has a linear variation. This means that the Errors in A and B and also the standard deviation in noise are exponential. The code is as follows:

```
#Plotting Aerr and Berr againt sigma in log scale for both axes
errorbar(sigma,A_error,A_error,fmt='ro',label='Aerr')
errorbar(sigma,B_error,B_error,fmt='go',label='Berr')
xscale('log')
yscale('log')
title('Q11: Variation of error with noise')
ylabel('Mean Squared Error $\longrightarrow$')
xlabel('\u03C3n $\longrightarrow$')
grid()
legend(loc='upper right')
show()
```

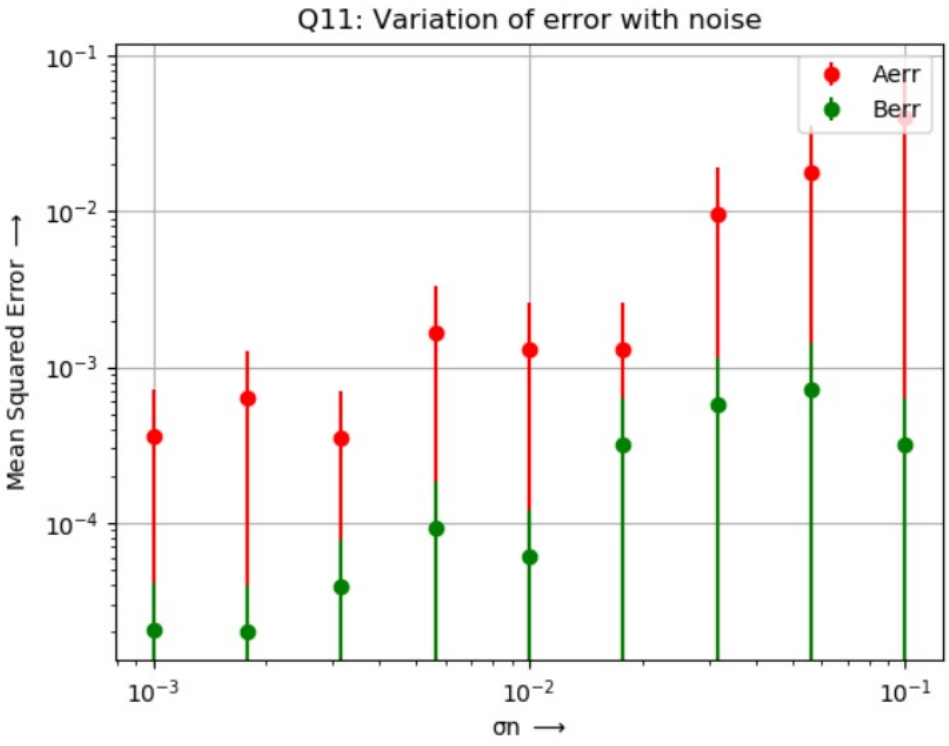


Figure 5: Errors in A and B vs Standard Deviation of noise in log scale

## 6 Conclusion

Through this assignment (Fitting Data to Models), we learned about the *scipy* and *matplotlib* libraries and how to make use of them in data analysis. The error plots show that error in the least-square solution increases with increase in deviation of noise. The variation of log of error in (A,B) with log of standard deviation of noise appears to be linear.