# EE2703: Assignment 3

Yogesh Agarwala
EE19B130

March 3, 2021

## 1 Introduction

This weeks Python assignment is mainly focused on studying the effect of noise on the fitting process. It starts off with generating data as a linear combination of the Bessel Function and $y = x$. Varying amounts of Noise is added to this combination.

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

where :
$J_2(t) = Bessel function$ and $n(t) = noise function$

What is noise? For us here, it is random fluctuations in the value due to many small random effects. The noise is given to be normally distributed, i.e., its probability distribution is given by

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

## 2 Assignment

### Q1. Data Generation

Data is generated by running the generate_data.py script. It utilizes the pylab library to find the value of the Bessel function and different values of time and then adds noise for 9 values of standard deviation which are uniformly sampled from a logarithmic scale. The data is written out to a file whose name is fitting.dat.

### Q2. Importing the Data

Numpy's loadtxt function is used to extract data from the 'fitting.dat' file. The data consists of 10 columns. The first column is time, while the remaining columns are data with varying noise levels.

## Q3. Plotting the Data

On plotting all the 9 columns of data (where the data columns correspond to the function with different amounts of noise added), the following graph was observed:
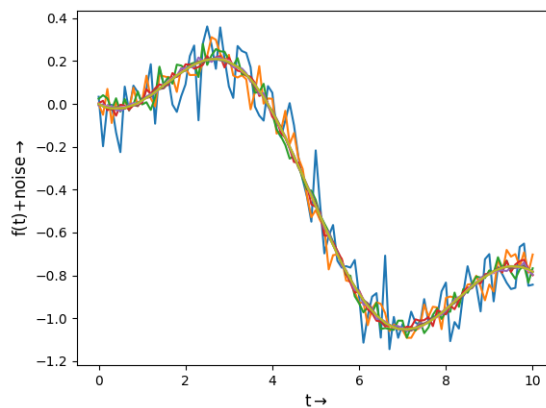


Figure 1: Varying Noise Levels

## Q4. Fitting a Function to this data

On plotting the original function $g(t, A, B) = AJ_2(t) + Bt$ where A = 1.05 and B = -0.105 the following graph is obtained:
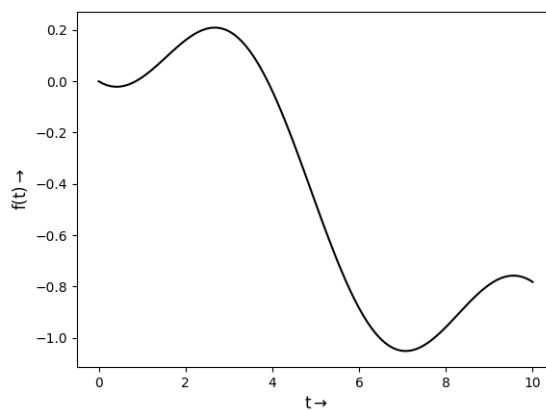


Figure 2: The Original function

So the combined Plot of Different Noise Levels and the Fitting Function is as follows:
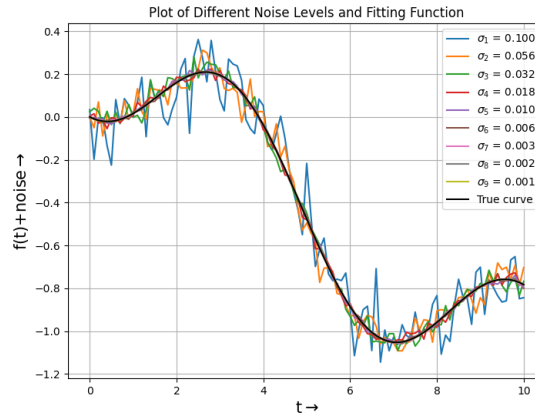


Figure 3: The Original function and Varying Noise Levels

## Q5. Plotting Error Bars

An errorbar is a convenient way of visualising the uncertainty in the reported measurement. The errorbars for the first data column (i.e corresponding to $\sigma = 0.1$) are plotted using the errorbar() function. The graph obtained by plotting every 5th data point with errorbars and the original data is as follows:
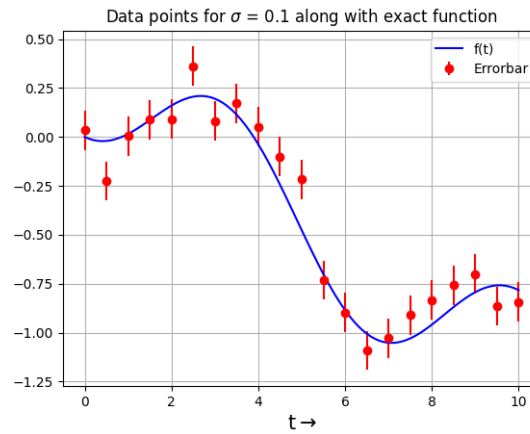


Figure 4: Error bars

3

## Q6. The Matrix Equation

In this part we have calculated g(t,A,B) using matrix multiplication as follows:

1: $g(t, A, B) = M. \begin{pmatrix} A_0 \\ B_0 \end{pmatrix}$ where M $= \begin{bmatrix} J_2(t_1) & t_1 \\ J_2(t_2) & t_2 \\ ... & ... \\ J_2(t_m) & t_m \end{bmatrix}$

2: $g(t, A_0, B_0)$

We can compare the results obtained using matrix multiplication and the user-defined function using the np.array equal() function, and it comes out that they are equal.

```
fn_column = sp.jn(2,time)
M = np.c_[fn_column,time] # construct M matrix

A = 1.05; B = -0.105
p = np.array([A,B]) # parameter matrix

G1 = np.matmul(M,p) # column vector obtained by multiplication
G2 = np.array(g(time,A,B)) # obtained directly from the function
```

## Q7. Mean Squared Error

We can find the MSE (mean square error) between our predicted A,B and Actual A,B using this formula:

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

where
$f_k = (k+1)^{th}$ column of the data

The following code snippet is used to compute this error e in each data column:

```
y_data = G1 #first column of data
y_actual = G2 #actual model

error = (np.square(y_data - y_actual)).mean()
print("Mean square error in calcutaion of M = ",error)
```

## Q8. Contour Plot of the Error

From the graph, the error function has one and only one minimum and it occurs at A = 1.00 and B = -0.10
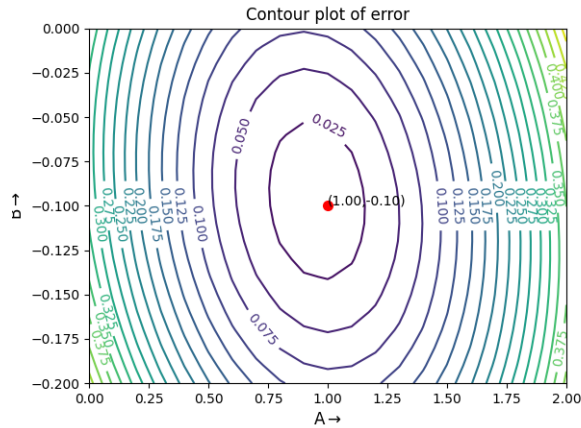


Figure 5: Contour Plot

## Q9. Best Estimate of A and B

AB is estimated by minimizing $|M * (AB) - b|$ where b = one of the columns of the data This can be done using the scipy.linalg.lstsq function as follows:

```python
def estimateAB(M,b):
        return np.linalg.lstsq(M,b,rcond=None)

def error_prediction(estimated, actual):
        A0,B0 = actual[0], actual[1]
        A1,B1 = estimated[0], estimated[1]
        return abs(A0-A1), abs(B0-B1)

# construct M matrix
fn_column = sp.jn(2,time)
M = np.c_[fn_column,time]

AB = [1.05, -0.105]
estimatedAB,_,_,_ = estimateAB(M,y[:,1]) #for first column data
print("Estimated_A,B__=_", list(estimatedAB))

errorA, errorB = error_prediction(estimatedAB,AB)
print("Error_in_A,B_=_", [errorA, errorB])
```

## Q10. Error in the estimation A and B on a linear Scale

The values of A and B are found out using scipy.linalg.lstsq in Python. Then the using the $M$ matrix given in the problem the values of the function are calculated. $f = M \begin{bmatrix} A \\ B \end{bmatrix}$ .The error in A and B from the original values of $A_0$ and $B_0$ is plotted below: Then the mean squared error values in $f(t)$ are calculated using the corresponding column in $fitting.dat$ and $f$.
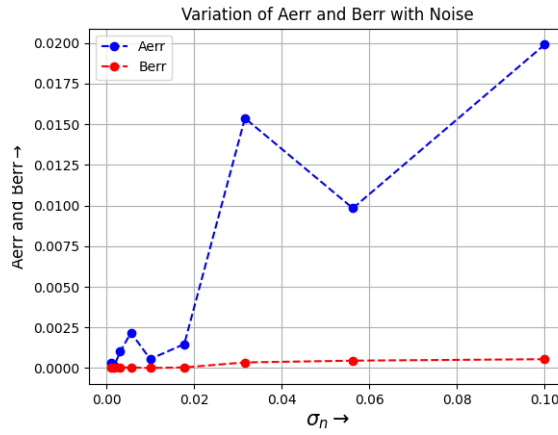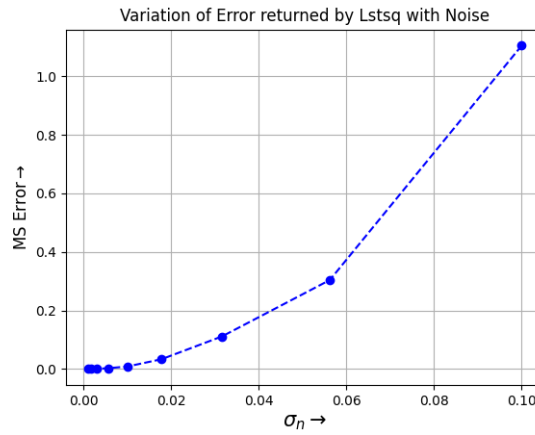


Figure 6: Error of A and B on a linear Scale



Figure 7: Error returned by lstsq on a linear Scale

6

## Q11. Error in the estimation A and B on a loglog Scale

Plotting this is similar to the previous part except for using the function plt.loglog instead of plt.plot.There is also an additional parameter to this function. i.e. basex. For convenience I have used basex = 10.
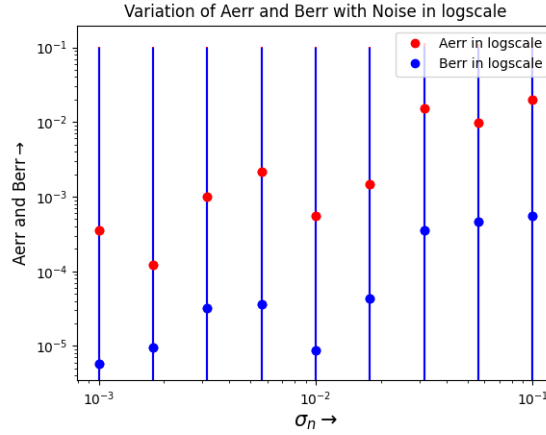


Figure 8: Error of A and B on a LogLog Scale

This graph is not Linear. However when we plot the error returned by the lstsq function It is nearly Linear This is because our noise varies on a logarithmic scale and it is expected that the error must also vary in a logarithmic manner because the error in the estimation is also based on a Gaussian distribution.
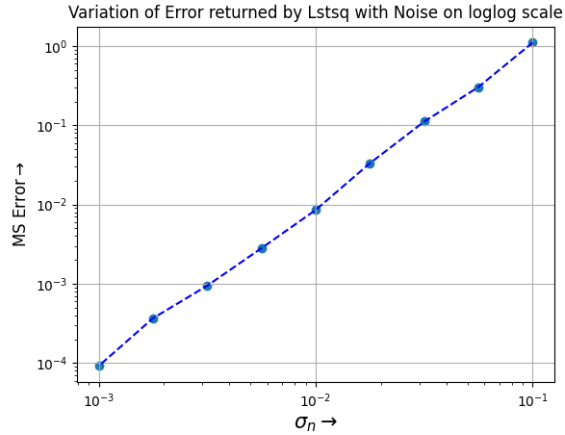


Figure 9: Error returned by lstsq on a LogLog Scale

# 3   Conclusion

The given noisy data was extracted and the best possible estimate for the underlying model parameters were found by minimizing the mean squared error. We have used the scipy library to estimate coefficients of a linear combination of 2 functions with varying amounts of noise. It was observed that the error returned by lstsq was approximately linear in the log scale, that is, as the noise in data used to estimate the linear combination increases, the error of prediction also increases linearly on a loglog scale.