Assignment No 3

Subhankar Chakraborty EE17B031

February 11, 2019

1 Extracting the data

The data consists of 10 columns, each of 101 rows with the first column being time and the rest being the Bessel function of the second order with different amounts of noise added. The **loadtext** function from **numpy** was used to load the data into different columns and then the data was plotted. The code below does the data extraction and it's suitable formatting.

```
data =np.loadtxt("fitting.dat")
columns ={}
for column_count in range(len(data[1,:])):
    columns[column_count] = data[:,column_count]
time = columns[0]
```

2 Plotting The data

The data corresponds to the second order Bessel function with different amounts of noise added.The different columns of the data correspond to:

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

Data was plotted using the plot function from **matplotlib**. The **errorbar** was used to indicate the error at each point.One in every 5 errors was plotted to make the plot readable.

```
for i in range(1,10):
    sigma = np.logspace(-1,-3,9).reshape(1,-1)
    plt.plot(time, standard,'gold')
    plt.errorbar(time[::5], columns[i][::5],sigma[0][0],fmt = 'o',capsize = 3)
    plt.xlabel('Time(t)')
    plt.ylabel(f"f{i}(t) = f(t)+n{i}(t)")
    plt.title("Noise and the True Value")
    plt.legend(["True Value",'Noise'])
```

```
plt.grid(True)
plt.show()
```

Let's visualize the data first.

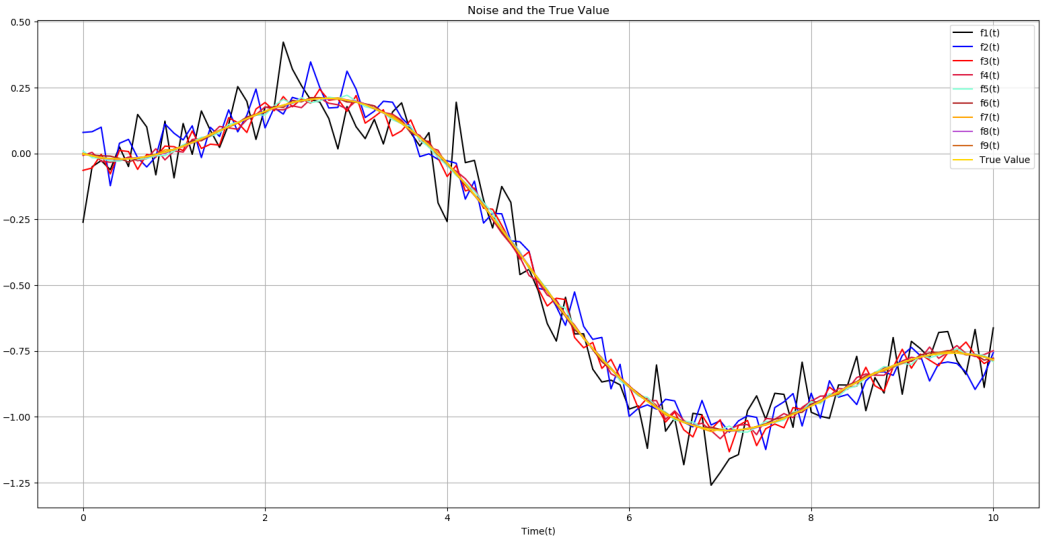

Figure 1: The different columns

The underlying curve is the second order Bessel function. Its graph looks as shown in Figure 2. The first column has the following shape with the errorbars indicating the deviation from the standard curve at every point.

3 Fitting a function the the data.

3.1 The Underlying structure

Each column has the underlying structure **g(t;A,B)** given by

$$g(t; A, B) = AJ_2(t) + Bt \tag{2}$$

Different columns will have different A's and B's depending on the noise. First we write a python code to generate *g(t)* given the value of A and B.

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

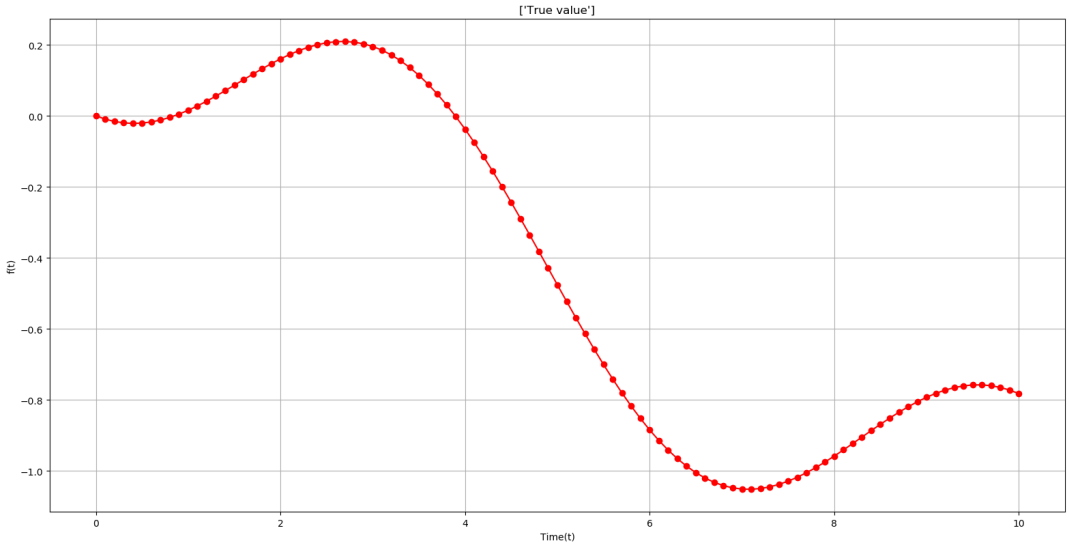

Figure 2: $\mathrm{f(t)} = 1.05\mathrm{J}_2(t) - 0.105(t)(1)$

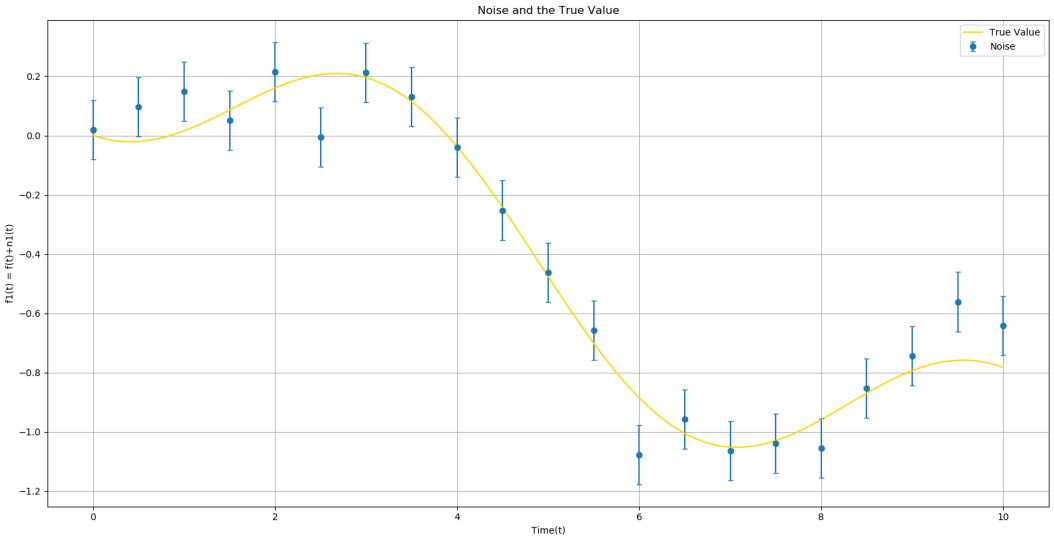

Figure 3: Column 1

3.2 Generating the Matrix to calculate g(t)

We can also generate the values of $g(t)$ by creating a matrix equation

$$g(t) = M * p \tag{3}$$

where **M** is a matrix whose first column is the second order Bessel function for the given values of time and the second column is the values of time.The matrix **p** is the column vector having values of A and b for which we have to generate g(t).

We can generate the matrix **M** quite easily.

```
def create_mat_M(t):
    time = np.asanyarray(t).reshape(-1,1)
    value = sp.jn(2,time)
    M=np.hstack((value,time))
    return(M)
```

4 Error values and plots

4.1 Creating the error matrix

We have a set of A's and a set of B's. A takes values in the range 0,0.1,0.2..,2 and B takes on values *-0.2,-0.19..,0*. There are 21*21 possible tuples of *(A.B)*. The mean squared error is defined as

$$\epsilon_{i,j} = \frac{1}{101}\sum_{k=0}^{100}(f_k - g(t_k, A_i, B_j))^2 \tag{4}$$

The code for the above is

```
#Making arrays for a and b.
a = np.arange(0,2.01,0.1)
b = np.arange(-0.2,0.001,0.01)
b[abs(b)<1e-5]=0
len_a = len(a)
len_b = len(b)

error_matrix = np.zeros((len_a,len_b))
for i in range(len_a):
    for j in range(len_b):
        error_matrix[i,j] =np.sum(((columns[1].reshape(-1,1)-
        g_using_matrix(time, a[i], b[j]))**2))/N
```

We get a 21*21 error matrix corresponding to different values of A's and B's.

4.2 Plotting the errors

We make a contour plot of the error matrix with respect to i and j. The code below does that.

```
i = np.arange(21)
j = np.arange(21)
levels = [0.03, 0.04, 0.06, 0.08, 0.10, 0.14, 0.18, 0.22, 0.26,0.31, 0.35]
ind = np.unravel_index(np.argmin(error_matrix, axis=None), error_matrix.shape)
temp =plt.contour(i,j,error_matrix,levels)
ind = np.asanyarray(ind)
plt.plot(ind[0], ind[1], marker = 'o', markersize = 3,color ='red')
plt.legend(["Minima"])
plt.clabel(temp, inline=True, fontsize=6)
plt.xlabel("i")
plt.ylabel('j')
plt.title("error_matrix[i,j] for f1(t)")
plt.colorbar()
plt.show()
```

The minima for the first column appears at (i,j) = (11,9).Figure 4 shows the contour plot.

4.3 Least Squared values

We use the *lstsq* function from *scipy.linalg* which estimates the least squared values for (A,B) corresponding to the 9 columns using the **M** matrix we have created.

```
M = create_mat_M(time)
A = {}
B = {}
for n in range(1,len(data[1,:])):
    [A[n], B[n]] , *rest_of_junk_produced = scipy.linalg.
    lstsq(M, columns[n].reshape(-1,1))
```

Now we plot the errors in A and B from their ideal values of (1.05, -0.105) depending on the noise value. First we plot errors with respect to sigma and then with respect to log(sigma).

```
#plotting against sigma
plt.stem((np.transpose(sigma)),np.asanyarray(error_a), 'y-o')
plt.stem((np.transpose(sigma)),np.asanyarray(error_b), 'k--')
plt.xlabel("Sigma")
plt.ylabel("Absolute Error")
plt.title("Error as a function of Standard Deviation")
plt.legend(['error_a', "error_b"])
plt.grid(True)
plt.show()
```

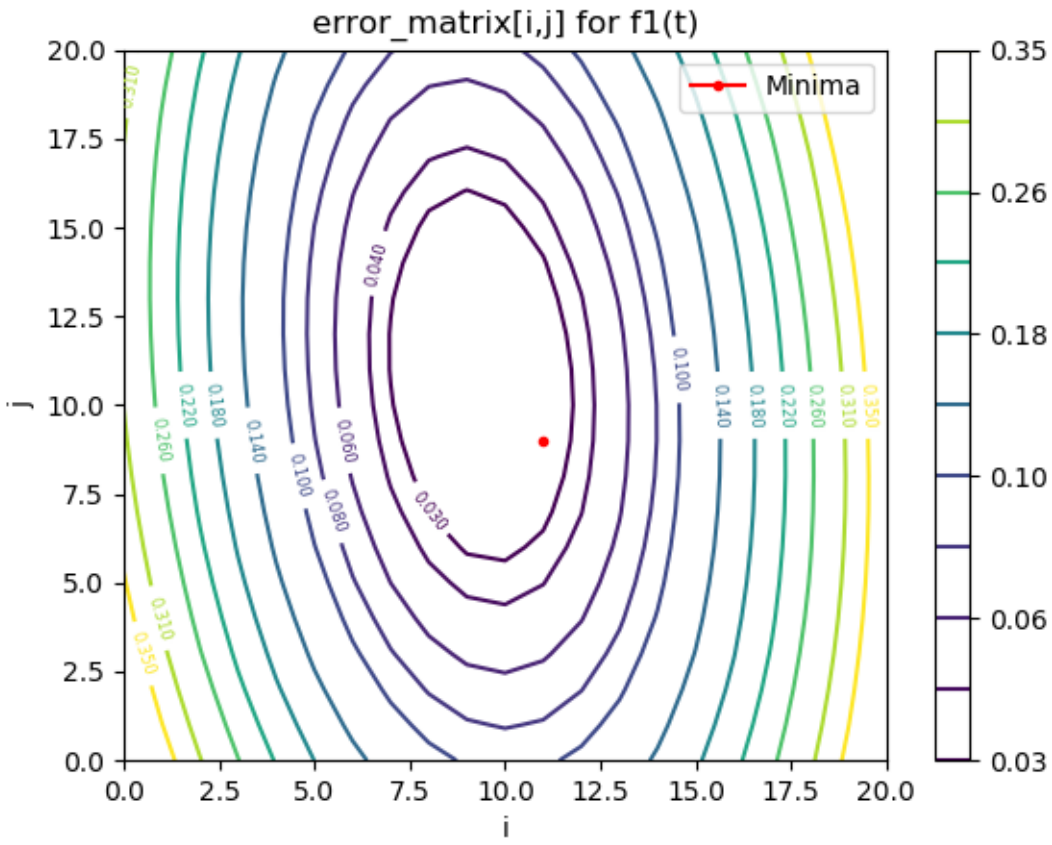

Figure 4: Contour Plot

```
#logarithmic plot
plt.stem((np.log(np.transpose(sigma))),np.asanyarray(error_a), 'y-o')
plt.yscale("log")
plt.stem(np.log((np.transpose(sigma))),np.asanyarray(error_b), 'k--')
plt.yscale("log")
plt.xlabel("Log(Sigma)")
plt.ylabel("Absolute Error")
plt.title("Error as a function of Standard Deviation")
plt.legend(['error_a', "error_b"])
plt.grid(True)
plt.show()
```

The plots generated for are shown in Figure 5 and figure 6.

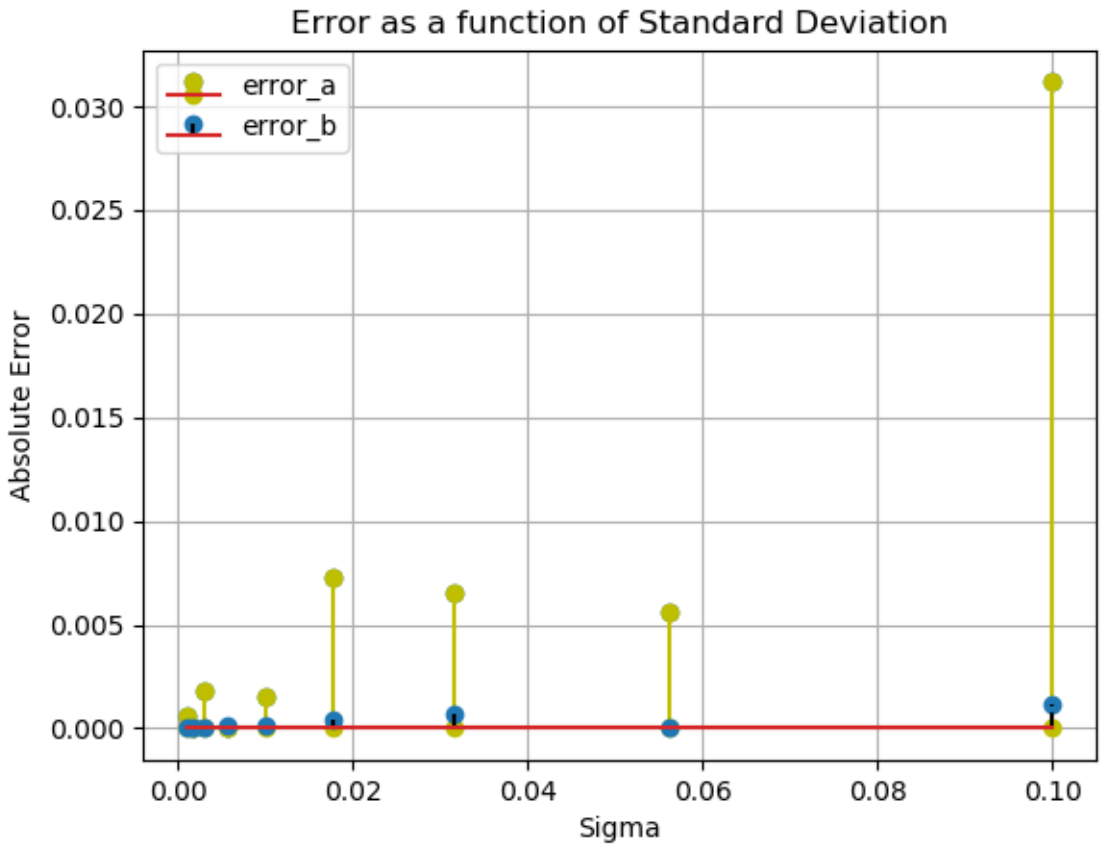

Figure 5: Error variation for sigma.

5 Inferences

The error in (A,B) when plotted with respect to *sigma*, show no sort of pattern. However when plotted with respect to *log(sigma)*, although not completely linear, shows a somehow linear trend. This means the errors in A and B roughly depend on some power of *sigma*.

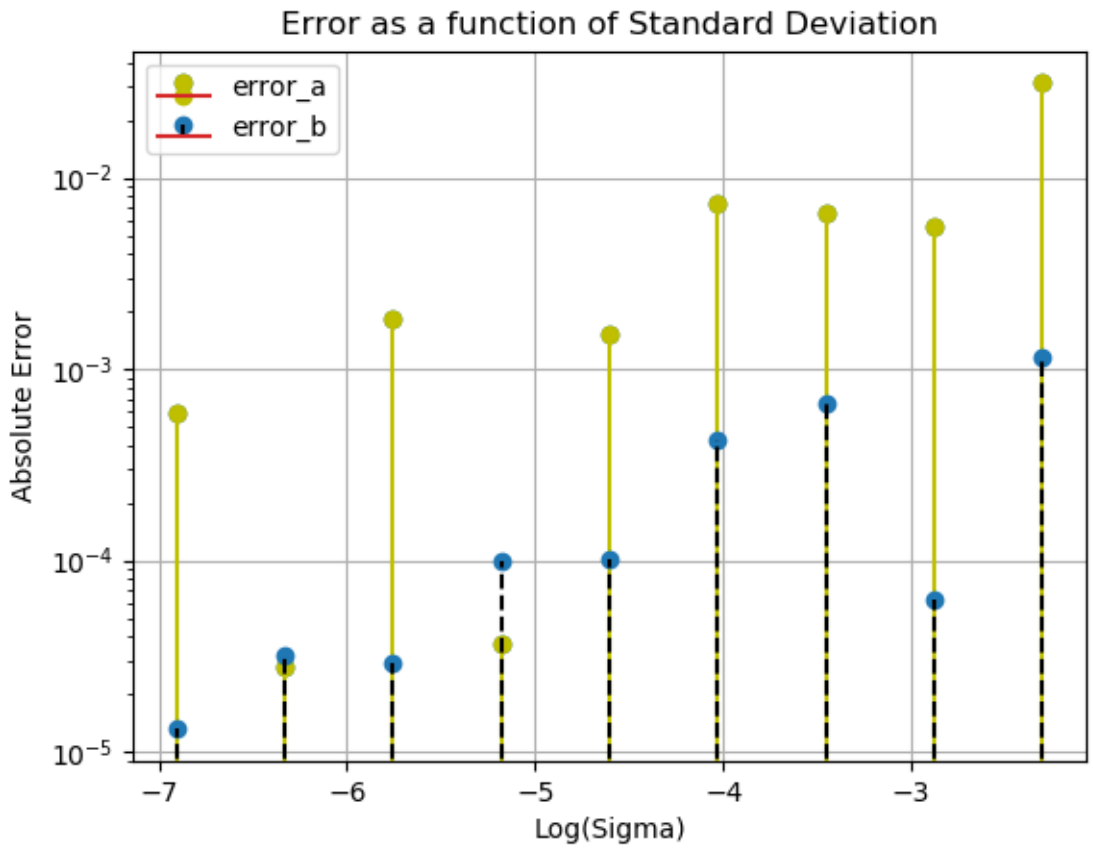

Figure 6: Error variation for sigma(Logarithmic).