Assignment 3

K.R.Srinivas - EE18B136

February 12, 2020

Abstract

This week's Python assignment focuses on the following topics.

- Reading data from file and parsing them.
- Analyzing the data and extracting information.
- Study the effect of noise on fitting process.
- Plot corresponding graph.

1 Introduction

The goal of this assignment is to estimate the error associated with the coefficient parameters A and B of the given function,and it's variation with different noise added to the input data.

While, the true value of the data is dictated by the equation [1],

$$f(t) = A.J(t) + Bt \tag{1}$$

noise $n(t)$ which is in general gaussian distributed with mean 1, adds to the measurements and we will also see that the error estimates in A and B are dependent on the standard deviation associated with $n(t)$.Thus, the data obtained can be modeled as follows.

$$f(t) = A.J(t) + Bt + n(t) \tag{2}$$

2 Assignment Tasks

2.1 Data Generation and Loading

The required data for our assignment is generated by running `generate_data.py` script.The program generates data using Bessel's function of second order "`jv(2,t)`" available in SciPy package and adds noise generated with help of `randn` function in numpy package.The generated data is then stored in

[1] $J(t)$ is the bessel's function evluated at t.

`.dat` format.

The data generated in the previous step is loaded into our mainscript using `loadtxt()` command.

```
try:
    data = loadtxt("fitting.dat")
except Exception:
    print("Data file unavailable!")
    exit()
```

The first coloumn of data corresponds to time instances at which the measurements are taken, and the remaining 9 coloumns corresponds to the noise added data. The standard deviation σ of the noise added to each coloumn is different and is known to be generated as

```
sigma = np.logspace(-1,-3,9)
```

2.2 Plotting Data

The actual data and noisy data is plotted using PyLab's `plot()` function to visualise the effect of noise on true data.

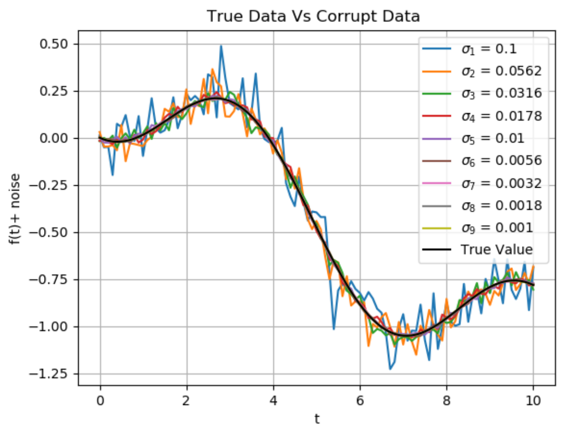

Figure 1: Visualising True Data And effect of Noise

The actual data is plotted using the following function.

```
def g(t,A=1.05,B=-0.105):return(A*sp.jv(2,t)+B*t)
```

In order to show how data diverges or varies from actual value we generate an error-bar plot at every $5th$ data item and compare with actual values.

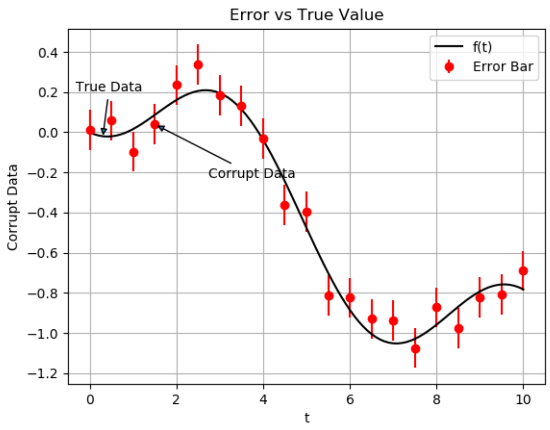

Figure 2: Error Bar Plot

In order to generate such plot we use the `errorbar()` function available in Pylab.

```
errorbar(data[::5],data[::5],stdev,fmt='ro')
stdev = sigma[0]
```

2.3 Matrix Generation

In our problem, the values of t are discrete and known (from the datafile). Obtain $g(t; A; B)$ as a column vector by creating a matrix equation:

$$g(t, A, B) = M * p$$

$$M = \begin{bmatrix} J_2(t_1) & t_1 \\ \vdots & \vdots \\ J_2(t_m) & t_m \end{bmatrix} p = \begin{bmatrix} A \\ B \end{bmatrix}$$

In order to verify if the matrix multiplication results with $A = 1.05$ and $B = -0.105$ is same as actual function `g(t,A,B)` we do the following.

```
M = c_[sp.jv(2, data[:, 0]), data[:, 0]]
p = c_[[A, B]]
G = dot(M, p)
G1 = np.array(g(data[:, 0],A,B))
print(np.array_equal(G,G1))
```

where the last command checks for equality using function `array_equal()` available in Numpy's package.

2.4 Error Calculation for Different values of Coefficients

Given the actual values of A and B, we calculate the Mean Square error for different values of A and B. We do it as follows

```
A_ = linspace(0, 2, 21)
B_ = linspace(-0.2, 0, 21)
error = zeros((len(A_), len(B_)))

for i in range(len(A_)):
    for j in range(len(B_)):
        error[i, j] = mean(square(g(data[:, 0], A_[i],
                                 B_[j])-g(data[:, 0])))
```

2.5 Contour Plot of Error

We generate a contour plot of the error and also check if it has a minimum.

```
CS = contour(A_, B_, error, levels=20)
xlabel("A")
ylabel("B")
title(r"Contour Plot of $\epsilon_{ij}$")
clabel(CS, inline=1, fontsize=10)
a = np.unravel_index(np.argmin(error[:,:]),error.shape)
plot(A_[a[0]],B_[a[1]],"ro")
```

PyLab offers the function `contour(X, Y, Z, [levels])` where X and Y are coordinates of Z and Z is the height values over which the contour is drawn. Level determines the number and positions of the contour lines / regions. While `argmin()` returns minimum value of the array passed to it, `unravel_index()` extracts the coressponding index in the array.

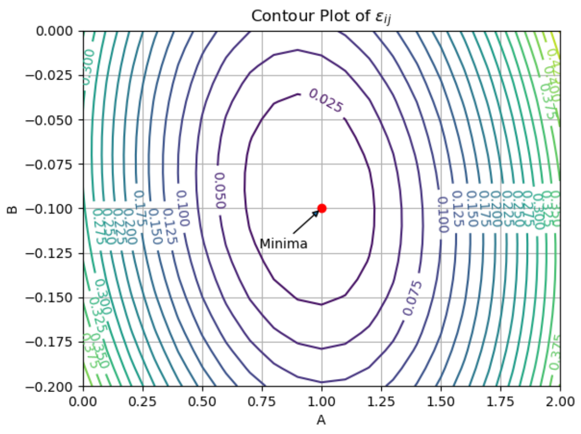

Figure 3: Contour Plot of Errors

2.6 Estimation of Errors on Different Scales

We use Python's `lstsq()` function available in SciPy package to solve $y = M * \begin{bmatrix} A \\ B \end{bmatrix}$, where y is the data values and estimate values for A and B. We then use `abs(x,y)` to evaluate the estimated error from actual values of A and B.

```
estimate = [lstsq(M, data[:, i], rcond = -1) [0] for i in
        range(1,10)]
estimate = np.asarray(estimate)
A_Error = square(abs(estimate[:,0]-np.ones(9)*A))
B_Error = square(abs(estimate[:,1]-np.ones(9)*B))
```

We now plot the errors in A and B in linear and loglog scale with respect to standard deviation in error (σ) as follows.

```
#LINEAR PLOT
plot(sigma,A_Error,'bo',label='A_Error')
plot(sigma,B_Error,'ro',label='B_Error')
ylabel('MS Error',fontsize=15)
xlabel(r'$\sigma_{n}$',fontsize=15)
legend()
title('Variation of Error with Noise')
grid()
show()
```

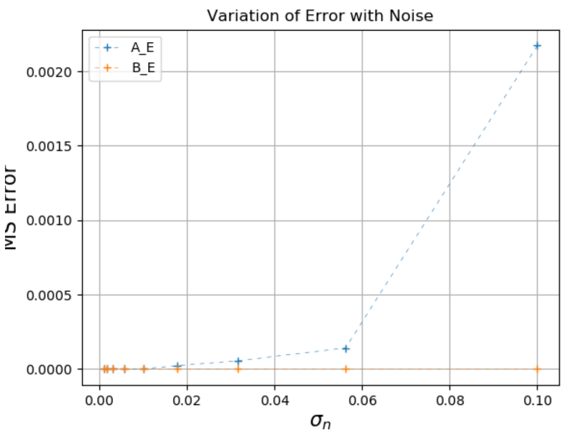

Figure 4: Variation of error with noise

```
#LOGLOG PLOT
loglog(sigma,A_Error,'bo',label = 'A_Error')
loglog(sigma,B_Error,'ro',label = 'B_Error')
legend()
errorbar(sigma, A_Error, std(A_Error), fmt='bo')
errorbar(sigma, B_Error, std(B_Error), fmt='ro')
ylabel('MS Error',fontsize=15)
xlabel(r'$\sigma_{n}$',fontsize=15)
title('Variation of Error with Noise')
grid()
show()
```

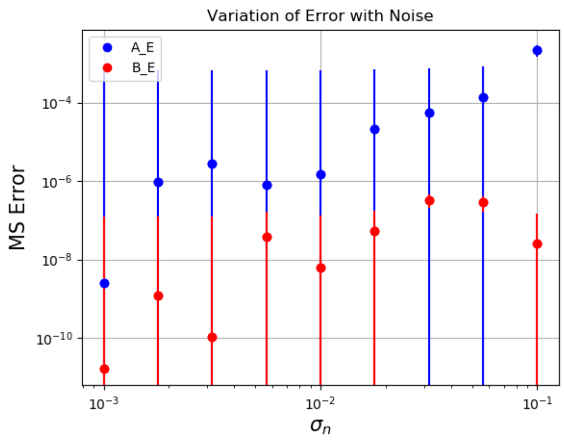

Figure 5: Variation of error with noise

3 Conclusion

Thus using the tools of SciPy and PyLab we have estimated the errors in coefficient of the function in the presence of noise.Also, we infer that the logarithm of noise linearly affects the logarithm of error.