Assignment No. 3

Srivenkat A (EE18B038)

February 11, 2020

1 Abstract

The assignment aims to

- parse a data file and extract the data
- study the effect of noise on a set of data
- plotting graphs using matplotlib or pylab

2 Introduction

The *generatedata.py* code given as a part of the assignment generates a data file with values of a function, derived from the Bessel function of the 2nd kind. It has sequences, each with varying amount of noise added. The assignment aims to analyse the effect of noise and plot corresponding graphs for the same.

3 Assignment

Task 1

The *generatedata.py* code was run and a data file *fitting.dat* was created. It has 10 columns, the 1st column being the time and subsequent columns being data points for the unknown function. Since a *randn* function is used to generate the noise, each time the code is run, it creates a new set of values.

Task 2

numpy.loadtxt() is used to extract the data from fitting.dat into a ndarray of dimensions 101*10.

Task 3

The data in the file is known to correspond to the function :

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

Where n(t) is a random amount of noise added. Each column of the data has a varying amount of noise added with a log-linearly varying standard deviation. The distribution of the noise for a particular std. deviation σ is given as:

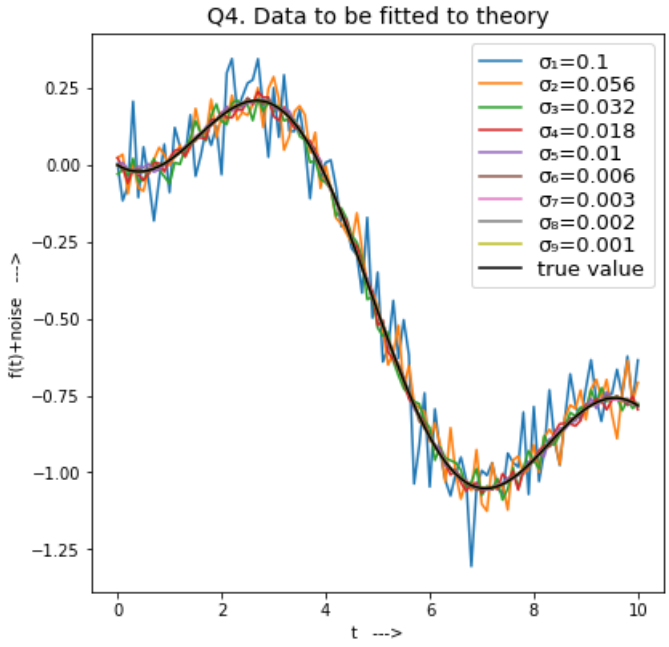
$$Pr(n(t)|\sigma) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{n(t)^2}{2\sigma^2})$$

Task 4

We consider the unknown function to be of the form:

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

We plot an estimated function with values of A and B as 1.05 and -0.105 and consider it to be the 'true value'. We also plot all the other columns with errors in the same graph.

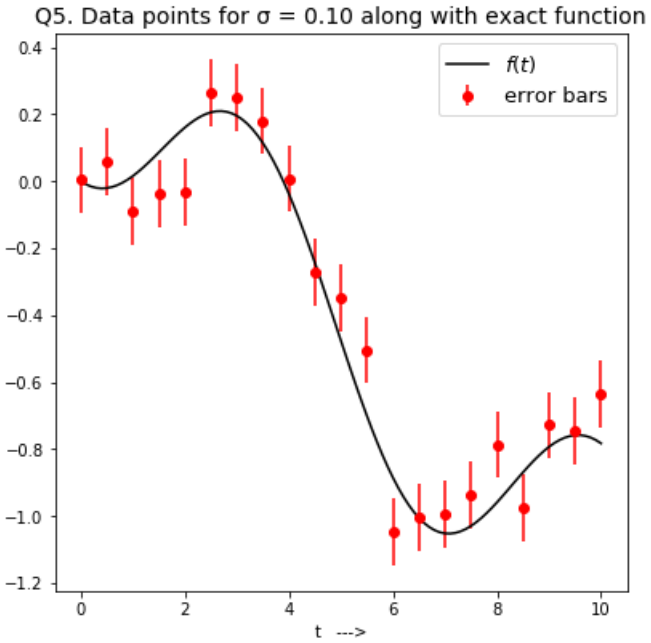


Each plot shows the same function with a different amount of noise added to it: It has its respective std. deviations as the label of the plot.

Task 5

To estimate how much the data diverges, we plot errorbars showing the standard deviation. For eg., we know that the !st column has an attached

noise with a std. deviation of 0.1. Its error bar will look like:



We plot the error bar for every 5th sample so that we get an comprehensible estimate for the entire set of data, (101 points).The height of the error is an estimate of the standard deviaton of the noise. For eg.,

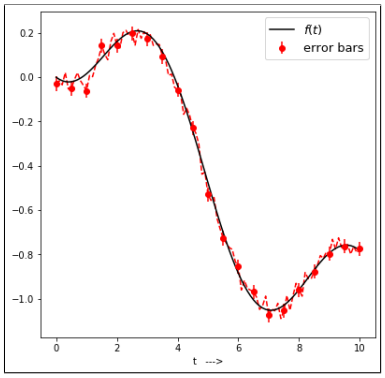


Figure 1: Noise std. dev of 0.032

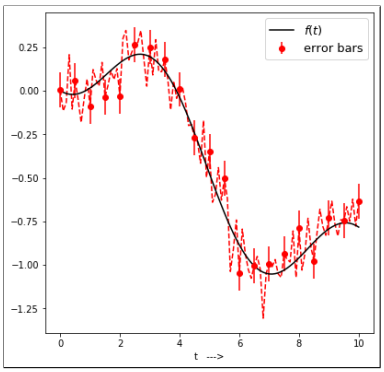


Figure 2: Noise std. dev of 0.1

From the above figure, it can be seen that the graph with a noise of lower std. dev more closely resembles the true value plot than the one with a more distributed noise.

Task 6

We form a coefficient matrix M of order 101x2 with J(t) and t as its columns. Use the **numpy.c_[col1,col2]** to form the matrix. Form the variable matrix X with A and B. Check if 1.05 and -0.105 give the true value by comparing the vector from multiplying M and X and evaluate its equality with the original true value matrix.

$$M. \begin{bmatrix} A_0 \\ B_0 \end{bmatrix} = g(t, A_0, B_0)$$

if (g==true_value).all(): command is used to check the equality of the 2 vectors. On running the code, they both are found to be equal and hereby consistent with our calculations.

Task 7

We consider all possible values of A and B in discrete steps, i.e, for A = 0,0.1,..., 2 and B = 0.2,0.19,...,0 and find the error in the estimation. Estimation error is calculated as:

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

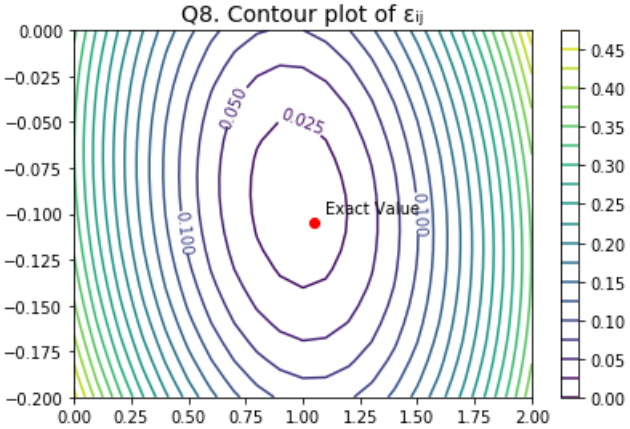
This error is called *mean squared error* and

sklearn.metrics.mean_squared_error(col_1,col_ref)

is the inbuilt function used to calculate it. It returns a 2d array of the error for possible A and B combinations.

Task 8

We plot the contour of the mean squared error obtained for different combinations of A and B in task 7. It is plotted over a 2d meshgrid of possible A and B values and its colour is a representation of the magnitude of the error.



In the contour plot, the darker coloured plots signify a lesser magnitude of error and the lighter plots signify an error of higher magnitude. So, the dark purple coloured contours signify a minimum. Hence as visible from the plot, multiple minimas are available.

Task 9

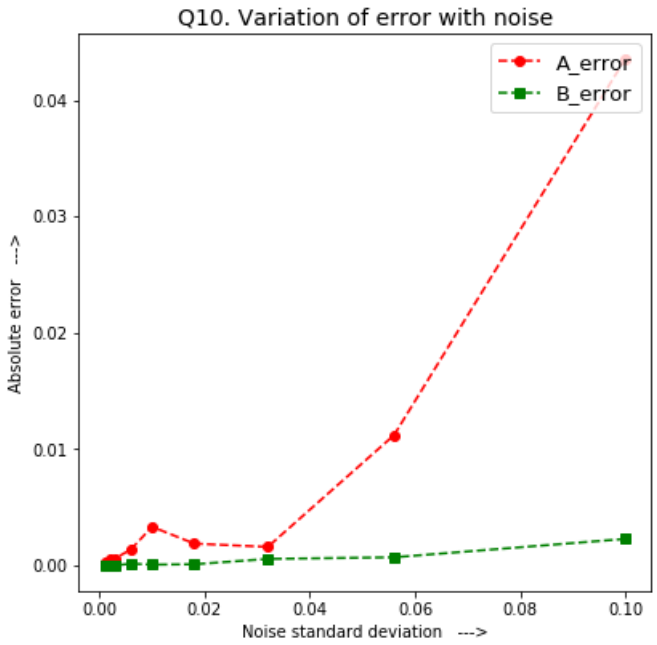
The least squares algorithm is conventionally used to predict a function that would best fit a given set of data. Here, we predict the best estimate of A and B for the matrix equation:

$$M. \begin{bmatrix} A_0 \\ B_0 \end{bmatrix} = data[i](t)$$

to hold true for different sets of data. we use the inbuilt function **scipy.linalg.lstsq()** to solve the equation.

Task 10

We extrapolate task 9 to apply the least squares function with all the data columns generated and return an array of optimal values of A and B. We then calculate the error in the estimation of A and B for a noisy data set and plot them to analyse trends in the the error estimates of A and B with the standard deviation of the noise added to the dataset.

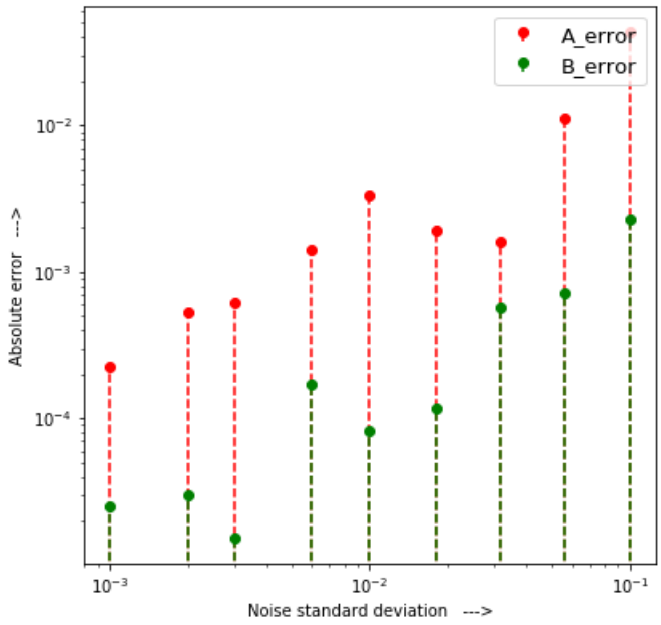


From the graph, we observe irregular trends in the error estimates of A and B with σ of noise. It signifies that the error in finding the coefficients of

the derived function is not directly proportional to the magnitude of noise added to the data set.

Task 11

To further find a relation between the error estimates of A, B and the std. deviation of noise added to the dataset, we plot the error graph in a log-log scale. The log-log plot is:



From this graph too, no direct algebraic relation can be predicted between error estimates of A, B and the std. deviation of noise added to the data set.

4 Results

To ensure that the conclusions we have arrived upon in this assignment are consistent, data sets with different noise distributions were analysed by running the *generatedata.py* code multiple times. The graphs for such an experiment are:

Variation of error in estimation of A and B with respect to the std.deviation of the noise added

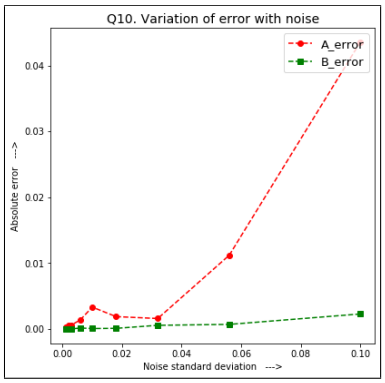


Figure 3: Data sample 2-linear scale

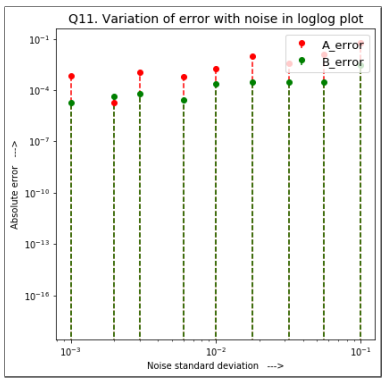


Figure 4: Data sample 2-log scale

Variation of error in estimation of A and B with std.deviation of noise added

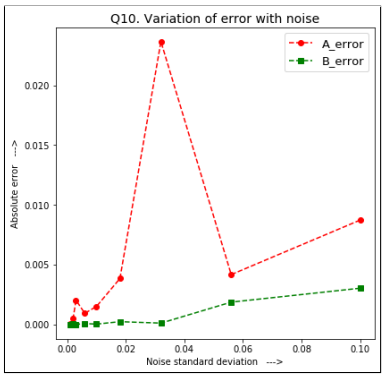


Figure 5: Data sample 3-linear scale

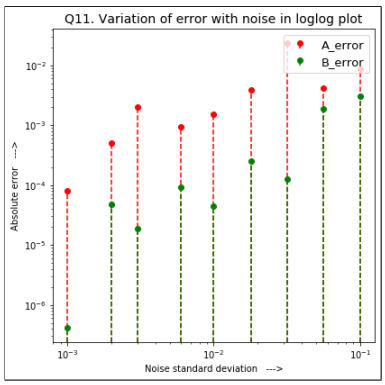


Figure 6: Data sample 3-log scale

This proves that **no definite algebraic relation can be deduced between the error in estimation of A, B and the standard deviation of the noise**, added to the data set.

5 Dealing with codes

The code used to generate the above plots is

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Feb  7 22:18:59 2020

@author: srivenkat
"""
```

```
import numpy as np
import matplotlib.pyplot as plt
import scipy
import scipy.special as sp
from sklearn.metrics import mean_squared_error

SIGMA = ' '      #declaring the greek symbols used
EPSILON = ' '
SUB = str.maketrans("0123456789", "                                    ")
#function to get subscript of a number

input_data = np.loadtxt('fitting.dat')       #extract data into numpy arra
data = np.transpose(input_data)      #getting data samples for a specific
time = data[0]

'''Q3: multiple data sequences vs time'''
sigma_num=np.round(np.logspace(-1,-3,9),3)
fig,graph1 = plt.subplots(figsize=(6.4,6.4))
for i in range(9):
    label_name = '{0}{1}'.format(SIGMA,i+1).translate(SUB)
    label_name = '{0}={1}'.format(label_name,sigma_num[i])
    graph1.plot(time,data[i+1],label=label_name)
graph1.set_xlabel('t    ---->')
graph1.set_ylabel('f(t)+noise    ---->')
graph1.set_title("Q4. Data to be fitted to theory",fontsize=14)

'''Q4: get true value for A=1.05, B=-0.105 '''
true_value=[]
def func_g(t,A,B):       #func to predict g vector for assumed coefficient
    g_mat = A*sp.jn(2,t) + B*t
    return g_mat
for i in range(len(time)):
    true_value.append(func_g(time[i],1.05,-0.105))
true_value = np.asarray(true_value)      #convert list to array
graph1.plot(time,true_value,'black',label='true value')
graph1.legend(loc='upper right', labelspacing=0.1, fontsize=13)
plt.show()

'''Q5: plot error bars for col 1,    =0.1'''
fig,graph2 = plt.subplots(figsize=(6.4,6.4))
graph2.set_title('Q5. Data points for %s = 0.10 along with exact function
graph2.errorbar(time[0::5],data[1][0::5],sigma_num[0],fmt='ro',label='err
#plot errorbars for downsampled array of column 1
graph2.plot(time,true_value,'black',label='$f(t)$')
```

```
graph2.set_xlabel('t    ---->')
graph2.legend(loc='upper right', labelspacing=0.5, fontsize=13)
plt.show()

'''Q6: check if function calculated discretely is same as assumed true va
def vect_func_g(A0,B0,t):        #func to return g vector for discrete tim
    data_col = sp.jn(2,t)
    time_col = np.transpose(t)
    M_mat = np.c_[data_col,time_col]    #to create matrix with column vec
    p_mat = np.array([A0,B0])
    g_mat = np.dot(M_mat,p_mat)
    return g_mat,M_mat
g_mat,M_mat = vect_func_g(1.05,-0.105,time)
if (g_mat == true_value).all():     #for comparing 2 vectors
    print("\nQ6. Both vectors are equal")
else:
    print("\nQ6. Both vectors are different")

'''Q7: to calculate mean sq. error for different A,B values'''
A_set = np.arange(0,2.1,0.1)        #array of possible A values
B_set = np.arange(-0.2,0.01,0.01)       #array of possible B values
mse = np.zeros(shape=[21,21])
col_ref = np.transpose(data[1])
for i in range(21):
    for j in range(21):
        g,temp = vect_func_g(A_set[i],B_set[j],time)
#return g vector for a specific A,B value
        mse[i][j] = mean_squared_error(g,col_ref)
#using scipy funcion to calc. mean sq. error

'''Q8: make a contour plot of errors and finding minimas for specific A,B
fig,graph3 = plt.subplots(figsize=(6.4,6.4))
cf = graph3.contour(A_set,B_set,mse,20)
graph3.plot(1.05,-0.105,'ro')
graph3.annotate("Exact Value",xy=(1.05,-0.105),xytext=(1.10,-0.100))
#to plot and label exact estimate
graph3.set_aspect(aspect=8)
graph3.set_title('Q8. Contour plot of%  s     ' % EPSILON,fontsize=14)
graph3.clabel(cf,[0,0.025,0.05,0.1],inline=1)       #marking specific lab
fig.colorbar(cf,shrink=0.66,extend='both')
plt.show()

'''Q9: finding best estimate for A,B'''
AB_soln = scipy.linalg.lstsq(M_mat,true_value)
```

```
AB_optimal = AB_soln[0]
print('\nQ9. Best estimate of A and B is:',AB_optimal)
#optimal value of A,B

'''Q10: calculating A and B for different datasets and plotting the error
fig,graph4 = plt.subplots(figsize=(6.4,6.4))
A_err = []
B_err = []
for i in range(len(data)-1):
    AB_collect = scipy.linalg.lstsq(M_mat,data[i+1])
#func to optimise M.x=P equation
    A_err.append(abs(AB_collect[0][0]-AB_optimal[0]))
    B_err.append(abs(AB_collect[0][1]-AB_optimal[1]))

graph4.plot(sigma_num,A_err,'r--o',label='A_error')
graph4.plot(sigma_num,B_err,'g--s',label='B_error')
graph4.set_xlabel('Noise standard deviation ---->')
graph4.set_ylabel('Absolute error ---->')
graph4.set_title('Q10. Variation of error with noise',fontsize=14)
graph4.legend(loc='upper right',fontsize=13)
plt.show()

'''Q11: analysing the A,B error estimates in loglog plots'''
fig,graph5 = plt.subplots(figsize=(6.4,6.4))
graph5.set_title('Q11. Variation of error with noise in loglog plot',font
graph5.stem(sigma_num,A_err,linefmt='r--',markerfmt='ro',basefmt='none',u
graph5.stem(sigma_num,B_err,linefmt='g--',markerfmt='go',basefmt='none',u
graph5.set_xscale('log')    #setting logarithmic scale in x and y axes
graph5.set_yscale('log')
graph5.set_xlabel('Noise standard deviation ---->')
graph5.set_ylabel('Absolute error ---->')
graph5.legend(loc='upper right',fontsize=13)
plt.show()
```