Assignment No 3: Report

-Akash Reddy A, EE17B001

February 13, 2019

1 Overview

Assignment No 3 uses Python to analyse sets of noisy data in the following manner:

- Reading (nine) noisy data sets and corresponding points of time from a file.
- Plotting the (nine) sets of noisy data generated versus time. The noise generated is normally distributed with given standard deviations.
- Plotting the noiseless function, and finding the best-fit curve from the assumed model(family of the noiseless function) for each of the noisy data sets, using Python command `lstsq` on a matrix equation.
- Plotting the errorbars of data points of a noisy data set.
- Plotting the contour plot of the *mean square error between the assumed model and the noisy data set*, and finding the point of minimum error.
- Plotting the error in the best-fit estimate as a function of the noise (standard deviation of the same).

2 Assignment 3

Plotting of the graphs is done mainly using matplotlib.pyplot (or pylab, which combines the pyplot functionality (for plotting) with the numpy functionality).

In our file `fitting.dat`, the first column is the values of time, while the other columns all correspond to the function:

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

where J_2(t) is the Bessel function of the first kind of order 2, and n(t) is the normally distributed noise added to the noiseless function, the distribution being:

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

where σ is the standard deviation of the distribution.

We now try to fit a function of the same general form:

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

to f(t). We create a matrix

$$M = \begin{bmatrix} J_2(\mathrm{t}_1) & t_1 \\ J_2(\mathrm{t}_2) & t_2 \\ \vdots & \\ J_2(\mathrm{t}_n) & t_n \end{bmatrix}$$

and solve the equation:

$$M \cdot \begin{pmatrix} A_0 \\ B_0 \end{pmatrix} = \begin{pmatrix} G_1 \\ G_2 \\ \vdots \\ G_n \end{pmatrix}$$

using the `scipy.linalg` function `lstsq`.

In codes, the `"uxxx"` escape characters are Unicode escape characters used to display Greek letters, subscripts or arrows on the plots.

This report tackles the Assignment in a Part-by-Part approach.

2.1 Part 1

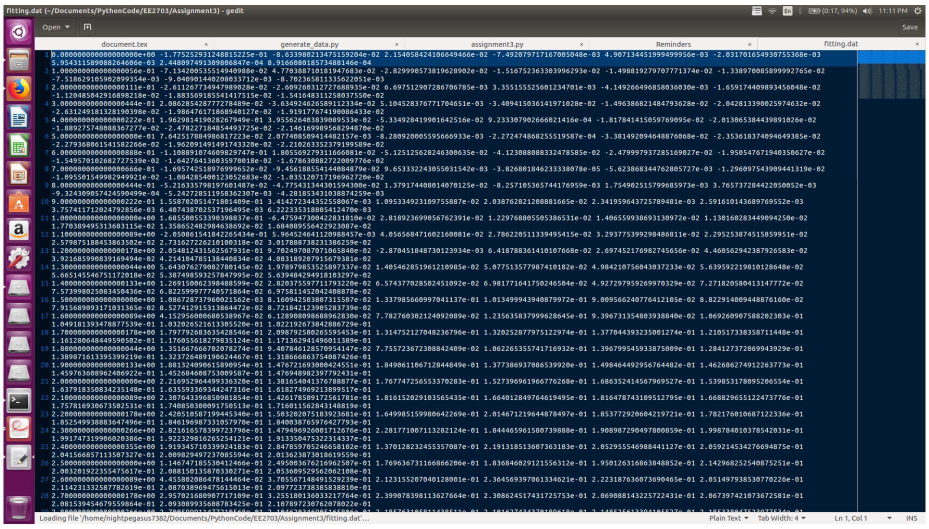
All that is needed to be done here is to run the generate_data.py file that we have, in order to generate `fitting.dat` as shown in Figure 1.

2.2 Part 2

Code used:

```
data=loadtxt("fitting.dat",dtype=float,comments="#")
time=[m[0] for m in data]
data=[[m[z] for m in data] for z in range(1,10)]
k=9
scl=logspace(-1, -3, k)
```

Upon first loading the data in `fitting.dat` into the array `data`, we then extract the first column into the array `time`. Then, we make `data` a list of lists, each list of which contains the data points of one column.



```
0.000000000000000000e+00 -1.775252931248815225e-01 -8.633980213475159204e-02 2.154058424106649466e-02 -7.492079717167005048e-03 4.907134451999499956e-03 -2.031701654930755368e-03 5.954311589088264606e-03 2.448097491309806847e-04 8.916608018573488146e-04
1.000000000000000056e-01 -7.134200535514940988e-02 4.770388710181947683e-02 -2.829990573819628902e-02 -1.516752363303996293e-02 -1.498819279707771374e-02 -1.338970085899992765e-02 -7.518629105902099354e-03 -9.040901440208033712e-03 -8.702365811335622051e-03
2.000000000000000111e-01 -2.611267734947989028e-02 -2.609260312727688935e-02 6.697512907286706785e-03 3.355155525601234701e-03 -4.149266496858036030e-03 -1.659174409893456048e-02 -1.120485042916898218e-02 -1.883569185541417515e-02 -1.541648311258037550e-02
3.000000000000000444e-01 2.086285428777278489e-02 -3.634924626589112334e-02 5.104528376771704651e-03 -3.409415036141971028e-02 -1.496386821484793628e-02 -2.042813390025974632e-02 -2.631249181328190398e-02 -1.986476171868940127e-02 -1.919177674190086433e-02
4.000000000000000222e-01 1.962981419028267949e-01 3.955626403839089533e-02 -5.334928419901642516e-02 9.233307902666021416e-04 -1.817841415059769095e-02 -2.013065384439891026e-02 -1.889275748088367277e-02 -2.478227184854493725e-02 -2.146169989568294870e-02
5.000000000000000000e-01 7.642517884986817223e-02 2.077408509414482157e-03 -8.280920005595666933e-03 -2.272474868255519587e-04 -3.381492094648876068e-02 -2.353618374094649385e-02 -2.279368061541582266e-02 -1.962091491491743320e-02 -2.210263352379199589e-02
6.000000000000000888e-01 -1.108891074609829747e-01 1.805569279311666081e-02 -5.125125628246300635e-02 -4.123088088332478585e-02 -2.479997937285169027e-02 -1.950547671940350627e-02 -1.549570102682727539e-02 -1.642764136035970018e-02 -1.678630882722009776e-02
7.000000000000000666e-01 -1.695742518976999652e-02 -9.456188554144084879e-02 9.653332243055031542e-03 -3.826801846233338078e-05 -5.623868344762805727e-03 -1.296097543909441319e-02 -1.095501549982949921e-02 -1.084285400123052683e-02 -1.035120717196962720e-02
8.000000000000000444e-01 -5.216335798197601487e-02 -4.775431344301594300e-02 1.379174408014070125e-02 -8.257105365744176959e-03 1.754902515799685973e-03 3.765737284422050052e-03 -9.324309057424590499e-04 -5.242728511958362307e-03 -4.281853431038874259e-03
9.000000000000000222e-01 1.558702051471801409e-01 3.414272344352558067e-03 1.095334923109755887e-02 2.038762821208881665e-02 2.341959643725789481e-03 2.591610143689769552e-03 3.757411712024792856e-03 6.407438702537196495e-03 6.222335318805412470e-03
1.000000000000000000e+00 1.685500553390398837e-01 -6.475947300422831010e-02 2.818923699056762391e-02 1.229768805505386531e-02 1.406559938693130972e-02 1.130160283449094250e-02 1.770389495313683115e-02 1.358652482984638692e-02 1.684089556422923087e-02
1.100000000000000089e+00 -2.050861541842265434e-01 5.964524641120988457e-03 4.056568471602160081e-02 2.786220511339495415e-02 3.293775399298486811e-02 2.295253874515859951e-02 2.579875188453863502e-02 2.731627226210100318e-02 3.017888738231386259e-02
1.200000000000000178e+00 2.054812431562567931e-01 9.702497087071065840e-02 -2.870451848730123934e-03 6.418788361410107668e-02 2.697452176982745656e-02 4.460562942387926583e-02 3.921685990839169494e-02 4.214104785138440834e-02 4.083189207915679381e-02
1.300000000000000044e+00 5.643076279082780145e-02 1.978979853525897337e-02 1.405462851961210985e-02 5.077513577987410182e-02 4.984210756043037233e-02 5.639592219810128648e-02 5.665145546751172018e-02 5.387498593257847995e-02 5.639484294918103297e-02
1.400000000000000133e+00 1.269150062398488599e-02 2.820375597711793220e-02 6.574377028502451092e-02 6.981771641750246504e-02 4.927297959269970329e-02 7.271820580413147772e-02 7.573998025083450436e-02 6.822599977740571864e-02 6.975811452042408878e-02
1.500000000000000000e+00 1.086728737960021562e-03 8.160942503807315507e-02 1.337985660997041137e-01 1.013499943940879972e-01 9.009566240776412105e-02 8.822914009448876160e-02 7.915689093171031365e-02 8.527412915313864472e-02 8.721842123905283739e-02
1.600000000000000089e+00 4.152956000680538967e-02 6.128908098688962830e-02 7.782760302124092089e-02 1.235635837999628645e-01 9.396731354803938840e-02 1.069260907588202303e-01 1.049181393478877539e-01 1.032026521613305520e-01 1.022192673842886729e-01
1.700000000000000178e+00 1.797792683635428546e-01 2.098792580265595453e-01 1.314752127048236796e-01 1.320252877975122974e-01 1.377044393235001274e-01 1.210517338358711448e-01 1.161280648449590502e-01 1.176055618279835124e-01 1.171362941496011389e-01
1.800000000000000044e+00 1.351667666702078274e-01 9.407846128570954147e-02 7.755723672308842409e-02 1.062265355741716932e-01 1.396799545933875009e-01 1.284127372069943929e-01 1.389871613395399219e-01 1.323726489190624467e-01 1.318666863754087426e-01
1.900000000000000133e+00 1.881324090615890954e-01 1.476721693000424551e-01 1.849061106712844849e-01 1.377386937086539920e-01 1.498464492956764482e-01 1.462686274912263773e-01 1.459763608962406922e-01 1.452684608753009587e-01 1.476948982397792431e-01
2.000000000000000000e+00 2.216952964499336320e-01 1.381654041376788877e-01 1.767747256553370283e-01 1.527396961966776268e-01 1.686352414567969527e-01 1.539853178095206554e-01 1.637918350834235148e-01 1.635593369344247316e-01 1.618274969213899517e-01
2.100000000000000089e+00 2.307643396850981854e-01 1.426178509172561781e-01 1.816152029103565435e-01 1.664012849764619495e-01 1.816478743109512795e-01 1.668829655122473776e-01 1.757816930673502531e-01 1.740850300091750513e-01 1.716011562843148819e-01
2.200000000000000178e+00 2.420510587199445340e-01 1.503202075183923681e-01 1.649985159980642269e-01 2.014671219644878497e-01 1.853772920604219721e-01 1.782176010687122336e-01 1.852549938883647496e-01 1.846196987331057970e-01 1.840038765976427793e-01
2.300000000000000266e+00 2.821616578399723796e-01 1.479496926001712676e-01 2.281771007113282124e-01 1.844465961580739888e-01 1.908987290497800859e-01 1.998784010378542031e-01 1.991747319906020386e-01 1.922329816265254121e-01 1.913350475322314337e-01
2.400000000000000355e+00 1.919345710339924183e-01 2.847859705246658102e-01 1.370128232455357087e-01 2.191318513607363183e-01 2.052955546988441127e-01 2.059214534276694875e-01 2.041566857113507327e-01 2.009829497237085594e-01 2.013623873018619559e-01
2.500000000000000000e+00 1.146747185530412466e-01 2.495003676216962507e-01 1.769636731166866206e-01 1.836846029121556312e-01 1.950126316863848852e-01 2.142968252540875251e-01 2.003201922355475617e-01 2.088150135870330271e-01 2.053609529562062108e-01
2.600000000000000089e+00 4.455802086478144464e-02 3.705567148491529239e-01 2.123155207040128001e-01 2.364569397061334621e-01 2.223187636073690465e-01 2.051497938530770226e-01 2.114231332587782619e-01 2.087038969475615013e-01 2.097723738385838810e-01
2.700000000000000178e+00 2.957021680907717109e-01 3.255180136033217764e-01 2.399078398113627664e-01 2.308624517431725753e-01 2.069088143225722431e-01 2.067397421073672581e-01 2.081539454679559864e-01 2.093089935608783425e-01 2.107897230762078022e-01
```

Figure 1: Screenshot of data points from fitting.dat

2.3 Part 3 and Part 4

Code used:

```
def g(t, A, B):
    return A*jn(2,t)+B*t
SUB=str.maketrans("0123456789","\u2080\u2081\u2082\u2083\
u2084\u2085\u2086\u2087\u2088\u2089")
for x in range(len(data)):
    plot(time, data[x], label= "\u03C3"+
    str(x+1).translate(SUB)+"=" +str(truncate(scl[x],4)))
plot(time,g(array(time), 1.05,-0.105),'-k',label = 'True Value')
legend(loc='upper right')
xlabel('t \u279d', size=15, fontstyle='italic')
ylabel('f(t) + noise \u279d', size=15, fontstyle='italic')
title(' Q4: Data to be fitted to theory')
grid(True)
show()
```

Plot generated:

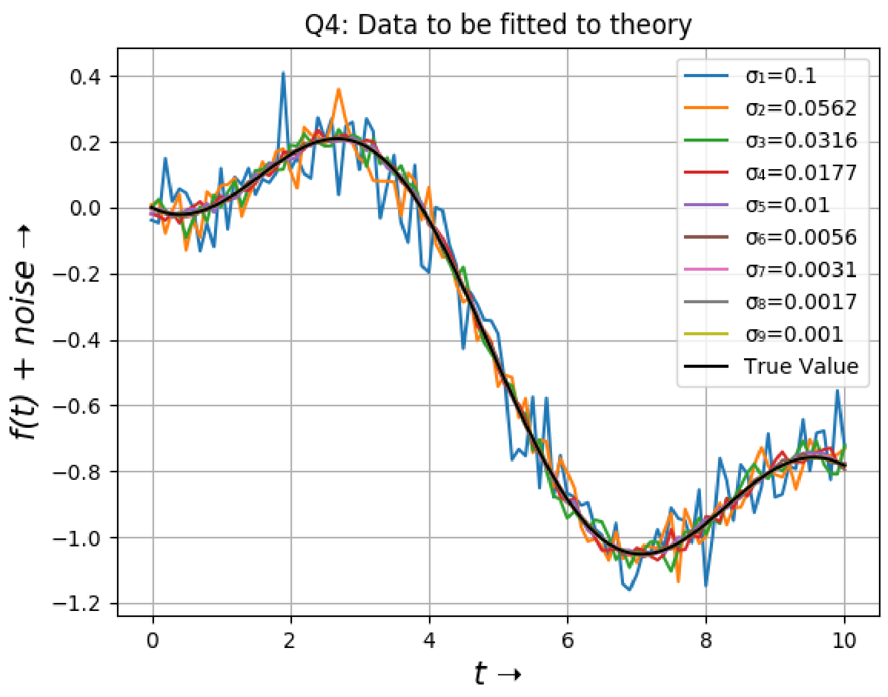


Figure 2: Plots of the 9 sets of noisy data and the noiseless function

The answers to Part 3 and Part 4 are clubbed because the curves are plotted on the same plot. We get a plot with time t on the x-axis and the 9 data sets f(t)+noise for Part 3. We also get the True Value (noiseless) curve in a very conspicuous black for Part 4.

The legend to the graph is visible at the upper right corner of the plot.

All of the above is seen in the plot in Figure 2.

2.4 Part 5

Code used:

```
errorbar(time[::5], data[0][::5], scl[0], fmt='ro', label =
'errorbar')
plot(time, g(array(time), 1.05,-0.105), '-k', label = 'f(t)')
legend(loc='upper right')
xlabel('t \u279d', size=15, fontstyle='italic')
title('Q5: Data points for \u03C3=0.10 along with exact function
f(t)')
grid(True)
```

```
show()
```

Plot generated:

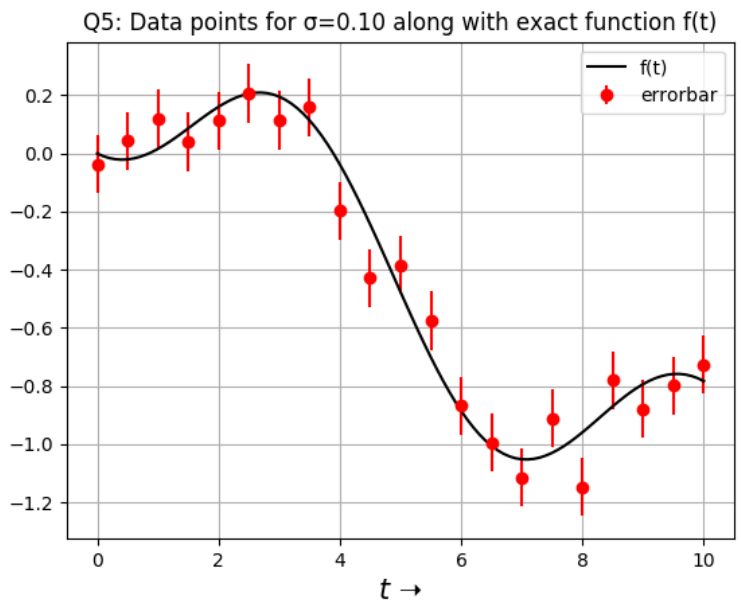


Figure 3: Plots of the errorbars at every fifth point, and the true function f(t)

An errorbar at a point on a function extends from the point to plus or minus the value of standard deviation of the function.

In order to plot the errorbars but let the graph be readable, we plot the errorbars for every **fifth** point instead of every point. The errorbars are plotted in red and the True Value function is plotted in black on the same plot. The legend to the graph is visible at the upper right corner of the plot. Time is on the x-axis.

All of the above is seen in the plot in Figure 3.

2.5 Part 6

Code used:

```
    J=array([jn(2, m) for m in time])
    T=array([m for m in time])
    M=c_[J,T]
```

The above code is used to construct the matrix M in the equation g(t,A,B) = M $\cdot p$.

In order to verify if

$$M\cdot \begin{pmatrix} A_0 \\ B_0 \end{pmatrix} = \texttt{g(t,}A_0\texttt{,}B_0\texttt{)}$$

we can select values of A_0 and B_0, generate an array with the two elements and solve for the matrix product using `numpy.matmul`. Subsequently, we can give A_0 and B_0 as inputs to g(t,A,B). Finally we check if the two resultant vectors are equal using `numpy.array_equal`.

2.6 Part 7 and Part 8

Code used:

```
Alist=arange(0,2.1,0.1)
Blist=arange(-0.20,0.01,0.01)
A,B=meshgrid(Alist, Blist)
epsilon=0
for k in range(len(time)):
    epsilon+=1/101*((data[0][k]-g(time[k], A,B))**2)
epsilon=transpose(epsilon)
plot(lstsq(M, data[0])[0][0], lstsq(M, data[0])[0][1], 'ro',
label="Exact location")
cp=contour(A,B, epsilon, 20)
clabel(cp, cp.levels[0:5], inline=True, fontsize=10)
xlabel('A \u279d', size=15, fontstyle='italic')
ylabel('B \u279d', size=15, fontstyle='italic')
title('Q8: Contour plot of \u03B5\u1D62\u2C7C')
show()
```

Plot generated:

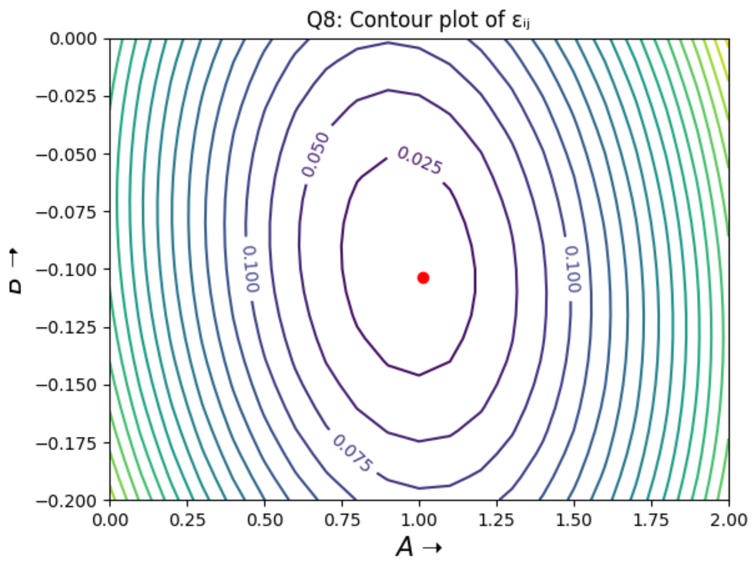


Figure 4: Contour Plot of ϵ_{ij}

Contours are lines drawn connecting points of equal function value, much like the contour lines in geographical maps that connect points at same height above sea level.

In Figure 4, we see the contour plot of the Mean Square Error between the first data column and the assumed model AJ$_2$(t)+Bt, which is calculated by:

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

where first column of `data` is used as f$_k$.

The contour plot has one minimum, which we have marked.

The red circle marks the exact location of the minimum of the contour plot, or the least Mean Square Error between g(t,A,B) and f. These values of A and B correspond to the best-fit curve to the data set that we have plotted the contour plot for.

2.7 Part 9 and Part 10

Code used:

```
G=[array(m) for m in data]
x=[lstsq(M, Gx) for Gx in G]
A_est=array([i[0][0] for i in x])
```

```
B_est=array([i[0][1] for i in x])
A_act=array([1.05 for i in range(len(x))])
B_act=array([-0.105 for i in range(len(x))])
plot(scl, abs(A_act-A_est), '--ro', label = 'Aerr', linewidth=0.5)
plot(scl, abs(B_act-B_est), '--go', label = 'Berr', linewidth=0.5)
legend(loc='upper right')
title('Q10: Variation of error with noise')
xlabel('Noise standard deviation \u279d', size=12, fontstyle='italic')
ylabel('MS error \u279d', size=12, fontstyle='italic')
grid(True)
show()
```

Plot generated:

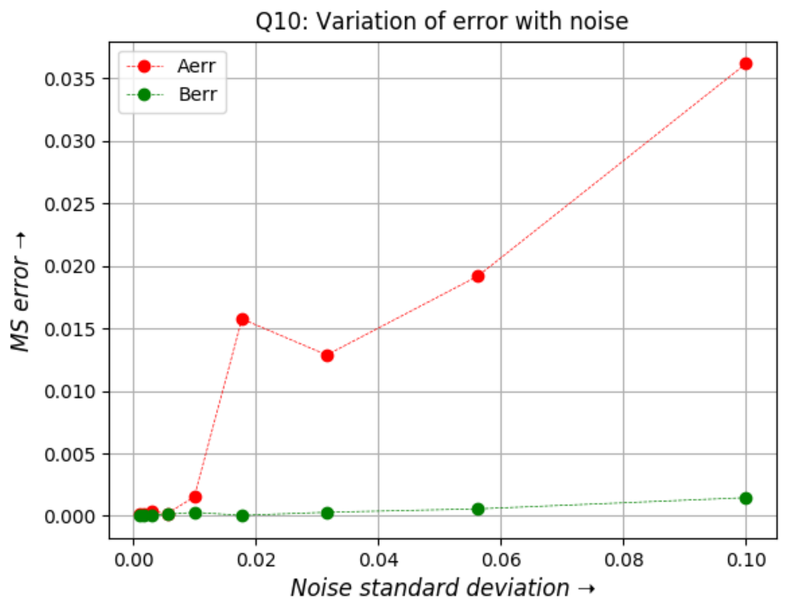


Figure 5: Variations of errors in estimated A and B with the standard deviation of the noise

Here, `lstsq` from `scipy.linalg` is used to solve the matrix equation Mx=G earlier obtained, where G is the set of noisy data points. M is the array we created earlier in Part 6. `lstsq` solves the equation Mx = G by computing a vector x that minimizes the Euclidean 2-norm $| b - ax |^2$. As a result, x is a vector that contains the values of A and B of the best-fit curve of the particular noisy data set we are solving. We then, in Figure 5, plot the errors in these estimates of A and B compared with the true values of the same, as a function of standard deviation of the noise σ.

There is a visible trend of linear growth of errors in estimates with standard deviation of the noise σ. We see that the A and B errors more or less imcrease with σ_n, except for a few anomalous points.

2.8 Part 11

Code used:

```
loglog(scl, abs(A_act-A_est), '--ro', label = 'Aerr', linewidth=0.5)
loglog(scl, abs(B_act-B_est), '--go', label = 'Berr', linewidth=0.5)
legend(loc='upper right')
title('Q11: Variation of error with noise (log-log plot)')
xlabel('\u03C3\u2099 \u279d', size=12, fontstyle='italic')
ylabel('MS error \u279d', size=12, fontstyle='italic')
grid(True)
show()
```

Plot generated:

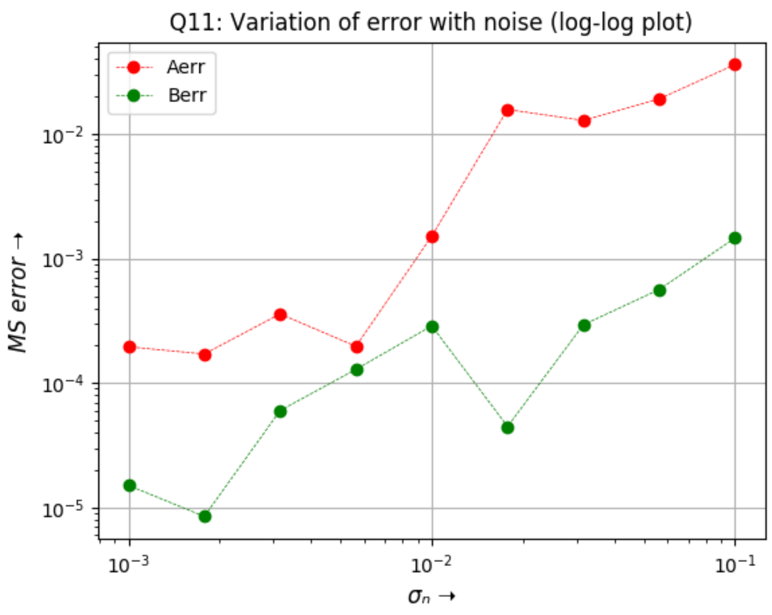


Figure 6: Variations of errors in estimated A and B with the standard deviation of the noise, plotted on log-log axes

The graph in Part 10 is re-plotted in Figure 6 albeit with log-log axes.

There is a visible trend of linear growth of errors in estimates with standard deviation of the noise σ. We see that the log values of A and B errors more or less increase with $\log(\sigma_n)$, except the plot is slightly erratic.

If

$$log(A_{err}) \propto log(\sigma_n)$$

[Linearity of the plot]

then,

$$\begin{aligned} log(A_{err}) &= klog(\sigma_n) \\ log(A_{err}) &= log(\sigma_n^k) \\ A_{err} &= \sigma_n^k \end{aligned}$$

and this follows for B$_{err}$ as well.

* *